PCT/JP03/07991

日本国特許庁
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

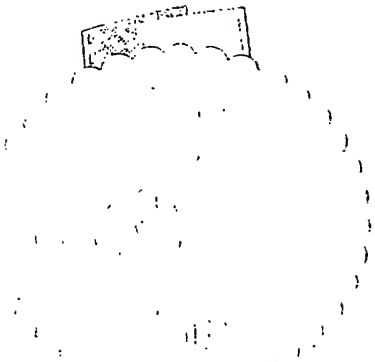
This is to certify that the annexed is a true copy of the following application as filed with this Office.

出願年月日
Date of Application: ２００２年　６月２６日

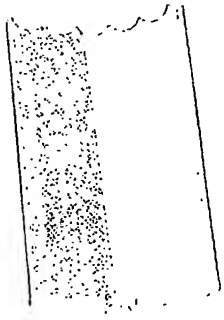
出願番号
Application Number: 特願２００２－１８６４７８

[ST. 10/C]: [ＪＰ２００２－１８６４７８]

出願人
Applicant(s): 株式会社イーツリーズ・ジャパン



PRIORITY DOCUMENT
SUBMITTED OR TRANSMITTED IN COMPLIANCE WITH RULE 17.1(a) OR (b)



２００３年　８月２１日

特許庁長官
Commissioner,
Japan Patent Office

今井康夫

出証番号　出証特２００３－３０６８０６５

【書類名】　特許願
【整理番号】　Q00603X0
【あて先】　特許庁長官　殿
【国際特許分類】　G06F　3/00
G06F 13/00
G06F 15/00
【発明の名称】　ソフトウェアをハードウェアに置き換えることにより通信プロトコルを高速処理する方法および装置
【請求項の数】　15
【発明者】
【住所又は居所】　神奈川県川崎市中原区小杉御殿町２－１１０
【氏名】　船田　悟史
【特許出願人】
【住所又は居所】　東京都渋谷区広尾１丁目１１番４号　共立ビル５０１
【氏名又は名称】　株式会社イーツリーズ・ジャパン
【代理人】
【識別番号】　100064447
【弁理士】
【氏名又は名称】　岡部　正夫
【選任した代理人】
【識別番号】　100085176
【弁理士】
【氏名又は名称】　加藤　伸晃
【選任した代理人】
【識別番号】　100106703
【弁理士】
【氏名又は名称】　産形　和央

【選任した代理人】

【識別番号】　100096943

【弁理士】

【氏名又は名称】　臼井　伸一

【選任した代理人】

【識別番号】　100091889

【弁理士】

【氏名又は名称】　藤野　育男

【選任した代理人】

【識別番号】　100101498

【弁理士】

【氏名又は名称】　越智　隆夫

【選任した代理人】

【識別番号】　100096688

【弁理士】

【氏名又は名称】　本宮　照久

【選任した代理人】

【識別番号】　100102808

【弁理士】

【氏名又は名称】　高梨　憲通

【選任した代理人】

【識別番号】　100104352

【弁理士】

【氏名又は名称】　朝日　伸光

【選任した代理人】

【識別番号】　100107401

【弁理士】

【氏名又は名称】　高橋　誠一郎

【選任した代理人】

【識別番号】　100106183

【弁理士】

【氏名又は名称】　吉澤　弘司

【手数料の表示】

【予納台帳番号】　013284

【納付金額】　21,000円

【提出物件の目録】

【物件名】　明細書　1

【物件名】　図面　1

【物件名】　要約書　1

【プルーフの要否】　要

【書類名】　明細書

【発明の名称】　ソフトウェアをハードウェアに置き換えることにより通信プロトコルを高速処理する方法および装置

【特許請求の範囲】

【請求項１】　ソフトウェアプログラムをハードウェア回路に変換するための方法であって、

所与のソフトウェアプログラムを１もしくは複数の機能単位に分割するステップと、

前記機能単位のそれぞれに対応する処理動作を行う複数のハードウェア処理回路モジュールを提供するステップと、

複数のデータセットの入力を１つの出力に統合するハードウェア合流回路モジュールを提供するステップと、

前記１もしくは複数の処理回路モジュールを相互に組み合わせ、もしくは、さらに前記合流回路モジュールと組み合わせて、前記ソフトウェアプログラムの全処理動作をハードウェア回路で実現できるようにするステップとを含む方法。

【請求項２】　ソフトウェアプログラムをハードウェア回路に変換するための方法であって、

所与のソフトウェアプログラムを１ないし複数の任意の機能単位に分割するステップと、

前記機能単位を任意の可変長のデータセットを介して連絡する０または１つの入力と、０もしくは１以上の出力とを有し、前記データセットに基づいて所定の処理動作を行うハードウェア処理回路モジュールを提供するステップと、

複数の前記データセットの入力を1つの出力に統合するハードウェア合流回路モジュールを提供するステップと、

１ないし複数の前記処理回路モジュールを相互に組み合わせ、もしくは、さらに１以上の前記合流回路モジュールのそれぞれの入・出力を組み合わせて、前記ソフトウェアプログラムの全処理動作をハードウェア回路で実現できるようにするステップとを含む方法。

【請求項３】　前記処理回路モジュールが、ゲートアレイから構成されてい

る請求項１又は２に記載の方法。

【請求項４】　前記データセットが、前記処理回路モジュールが直接または間接的に接続される他の前記処理回路モジュールを制御する情報を有するヘッダ部分と、データ部分とを含む請求項１又は２に記載の方法。

【請求項５】　前記合流回路モジュールが、一時的に入力された前記データセットを保存するための記憶装置を含む請求項１又は２に記載の方法。

【請求項６】　前記処理回路モジュールおよび前記合流回路モジュールが、入力された前記データセットの処理状態をフィードバックするための手段を含む請求項１又は２に記載の方法。

【請求項７】　ソフトウェアプログラムをハードウェア回路に変換するための方法であって、

所与のソフトウェアプログラムを１もしくは複数の機能単位に分割するステップと、

前記分割された機能単位の中からハードウェア化する機能単位を選択するステップと、

前記選択された機能単位のそれぞれに対応する処理動作を行う複数のハードウェア処理回路モジュールを提供するステップと、

複数のデータセットの入力を１つの出力に統合するハードウェア合流回路モジュールを提供するステップと、

前記１もしくは複数の処理回路モジュールを相互に組み合わせ、もしくは、さらに前記合流回路モジュールと組み合わせて、前記ソフトウェアプログラムの全処理動作をハードウェア回路と汎用計算機によるソフトウェア部分との組み合わせで実現できるようにするステップとを含む方法。

【請求項８】　前記処理回路モジュール又は合流回路モジュールをソフトウェア部分と組み合わせるステップにおいて、前記ソフトウェアプログラムの置換されるべき機能単位は動作検証されながら１つずつ前記処理回路モジュール又は合流回路モジュールに置換される、請求項７に記載の方法。

【請求項９】　請求項１ないし８のいずれかに記載の方法によって製作された回路を含む装置。

【請求項１０】　情報処理装置であって、

所与の情報処理用ソフトウェアを、その機能単位毎に、ハードウェア化することにより構成される複数のハードウェアモジュールと、

該複数のハードウェアモジュール間で、データをデータセットの単位で、かつ単方向に伝送するための伝送手段とを含む情報処理装置。

【請求項１１】　前記ソフトウェアの機能単位が、例えば、Ｃ言語の関数呼び出しによって、ヘッダ・データ情報のやり取りおよび処理が行われるようなソフトウェア要素に対応するものである請求項１０に記載の情報処理装置。

【請求項１２】　ハードウェアによる情報処理装置であって、

各々が、情報データの入力に応答して、特定の情報処理を実行する複数の処理回路モジュールと、

複数系統の情報データを１つの系統に合流させる合流回路モジュールと、

前記処理回路モジュール間および前記合流回路モジュール間、並びに前記処理回路モジュールと前記合流モジュールの間で、単方向にて前記情報データをデータセットの形式で送受するための伝送手段とを含むことを特徴とする情報処理装置。

【請求項１３】　１ないし複数の処理回路モジュールと、０もしくは１以上の合流回路モジュールとを含むハードウェアによる情報処理装置であって、

前記処理回路モジュールおよび前記合流回路モジュールは、所定の情報を含むデータセットによる単一方向の情報伝達を、前記処理回路モジュール、前記合流回路モジュールまたはＩ／Ｏインターフェースとの間で行い、

前記処理回路モジュールの各々は、そのモジュール固有の機能を果たす回路を有するとともに、０または１の入力と、０以上の出力とを有し、

前記合流回路モジュールは、２つ以上の入力および１つの出力を有し、２つ以上の回路から出力されるデータセットを１つの出力に合流させる処理を行うことを特徴とする情報処理装置。

【請求項１４】　インターネットサーバの機能を有する、請求項１０ないし１３のいずれかに記載の情報処理装置。

【請求項１５】　データマイニング、自然言語処理、ネットワーク情報処理

、ＤＮＡ計算シミュレータ、物理シミュレーション、および音声・画像処理の機能のうちの一機能有する、請求項１０ないし１３のいずれかに記載の情報処理装置。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】

本発明は、情報の伝達および計算処理、特にインターネットサーバ機能をハードウェアモジュールによって実現する方法および装置に関する。さらに、本発明は、メモリ・入力／出力装置・中央処理ユニット間の情報・他の信号の相互接続・転送を、統一アクセス制御手法を用いてＷｅｂサーバ、メールサーバ、ＦＴＰサーバおよびＤＮＳサーバなどをハードウェアで実現することを可能とする方法および装置に関する。

【０００２】

【従来の技術】

従来は、データマイニング、自然言語処理、ネットワーク情報処理（Ｗｅｂによる情報提供、アプリケーション処理、情報検索）、ＤＮＡ計算シミュレータ、および物理シミュレーション（新素材の特性解析、たんぱく質構造予測、惑星軌道予測など）、および音声・画像処理（リアルタイム圧縮・展開）、のような様々な情報の伝達および計算処理は、ＣＰＵを１つまたは複数個使った汎用的な計算機（コンピュータ）と、専用のソフトウェアとを利用して一貫した処理を行うことによって行われてきた。

【０００３】

【発明が解決しようとする課題】

従来の汎用的なＣＰＵ計算機を用いて様々な情報の伝達および計算処理を行う場合、命令を読み込まないと次の処理が分からない、処理結果によって次の命令が変化する（前もってデータを準備することができない）、あるいはデータをメモリから読み込まないと処理が進まない（データの読み込み速度が処理速度を決定する）といった、処理内容の汎用性を重視しすぎることに起因して処理速度が低下し、必要な処理速度が得られないという問題があった。

【０００４】

ここで、従来のＣＰＵ計算機に依存する方法を用いたインターネットサーバを例にあげると、サーバに対してユーザからのアクセスが集中すると、計算機はユーザのリクエストを処理しきれずにサーバはその機能を停止し、インターネットサービスが利用できなくなる事態が生じる。結果として、インターネットインフラに対しての信用が低下することとなる。また、今後のＡＤＳＬ・ＳＤＳＬおよび光ケーブルなどの拡充にともない、サーバに対する負荷が飛躍的に増大することが予想されている。

【０００５】

一方、上記問題を回避するために、専用計算機を利用すると都合が良いことが知られている。専用計算機はアプリケーションに応じた専用の計算回路を持っているため、専用計算機を用いて様々な情報の伝達および計算処理を行うと、汎用的なＣＰＵ計算機のような処理速度の低下を生ずることなく、所望の処理速度を得ることができる。

【０００６】

しかしながら、専用計算機を採用する場合、アプリケーションに応じた専用の計算回路を設計、製造する必要があるので、膨大な設計時間、製造コストなどがそのアプリケーションごとにかかってしまうという問題点があった。

【０００７】

図１は、このような専用計算機の回路を設計する際の問題点を端的に示すものである。図１に示されるように、一般に、一つの回路は複数の入出力を持ち、複数の信号を同時に処理している。そのため、回路内での入力値（ａ，ｂ，ｃ，ｄ）間、出力値（ｗ，ｘ，ｙ，ｚ）間、および入出力値間のタイミングは複雑な様相を呈し、複数の回路の連携に至っては、さらにタイミングが複雑となる。したがって、それぞれの複雑性に関連した入出力のタイミングを制御することは非常に困難であるため、その回路を設計することは至難の技であった。

【０００８】

また、従来、ハードウェアとソフトウェアからなる装置の開発は、装置仕様に従って、まず、ハードウェアの分担する機能とソフトウェアの分担する機能とに

分割し、つづいて仕様に基づいてハードウェア部分とソフトウェア部分とがそれぞれ同時にかつ個別に開発され、最後にハードウェア部分とソフトウェア部分とが統合されて動作検証するというように行われてきた。以上のような順序で開発が進められるので、ハードウェア（あるいはソフトウェア）が出来上がるまでソフトウェア（あるいはハードウェア）のデバッグができないという問題があった。この問題を解決するためにハードウェア（あるいはソフトウェア）をシミュレートするソフトウェア（あるいはハードウェア）を作って、ソフト（あるいはハード）のデバッグを行ってきたが、シミュレートするソフトウェアやハードウェアを作る工程を必要とし、２度手間となっていた。

【０００９】

本発明は、上記従来技術の問題点に鑑みなされたものであり、処理内容の弾力性（汎用性）を担保するとともに、高速処理が可能で、かつ比較的小さな回路規模で情報伝達処理装置を実現する技術手法を提供することを目的とする。

【００１０】

【課題を解決するための手段】

上記目的を達成するために、本発明は、以下に詳述するようなソフトウェアをハードウェアモジュールに変換せしめるための方法と、この方法を用いて構成されるハードウェアモジュールを用いた情報処理装置とを提供する。

【００１１】

本発明のソフトウェアーハードウェア変換方法は、所与のソフトウェアプログラムをハードウェア回路に変換するために、基本的に、ソフトウェアプログラムを複数の機能単位に分割するステップと、機能単位のそれぞれに対応する処理動作を行う複数の処理回路モジュールを提供するステップと、複数のデータセットの入力を１つの出力に統合する合流回路モジュールを提供するステップと、複数の処理回路モジュールを組み合わせ、もしくは、さらに合流回路モジュールと組み合わせて、ソフトウェアプログラムの全処理動作をハードウェア回路で実現できるようにするステップとを含む。また、所定の機能単位だけをハードウェア化して汎用の計算機によるソフトウェア部分とハードウェア化を共存させることも可能である。

【００１２】

より具体的には、本発明の方法は、所与のソフトウェアプログラムを１ないし複数の任意の機能単位に分割するステップと、機能単位を任意の可変長のデータセットを介して連絡する０または１つの入力と０もしくは１以上の出力を有し、前記データセットに基づいて所定の処理動作を行うハードウェア処理回路モジュールを提供するステップと、複数のデータセットの入力を１つの出力に統合する合流回路モジュールを提供するステップと１ないし複数の処理回路モジュールを相互に組み合わせ、もしくはさらに１以上の合流回路モジュールのそれぞれの入・出力を組み合わせて、ソフトウェアプログラムの処理動作を前記ハードウェア回路で実現できるようにするステップとを含む。

【００１３】

また、本発明のハードウェアモジュールを用いた情報処理装置は、基本的に、所与の情報処理用ソフトウェアを、その機能単位毎に、ハードウェア化することにより構成される複数のハードウェアモジュールと、複数のハードウェアモジュール間で、データをデータセット単位で、かつ単方向に伝送するための信号伝送手段とから構成される。そして、典型的には、ソフトウェアの機能単位は、例えば、Ｃ言語の関数呼び出しによって、ヘッダ・データ情報のやり取りと、処理とが行われるようなソフトウェア要素に対応するものとなっている。

【００１４】

より具体的には、本発明の装置は、ハードウェアによる情報処理を行うために、１ないし複数の処理回路モジュールと、０もしくは１以上の合流回路モジュールと、から基本的に構成され、処理回路モジュールと合流回路モジュールが所定の情報を含むパケットによる単一方向の情報伝達を、処理回路モジュール、合流回路モジュールまたはＩ／Ｏインターフェースと行い、処理回路モジュールのそれぞれが、それぞれに所定の機能を果たす回路を有し、および０または１の入力と０以上の出力を有し、合流回路モジュールは、２つ以上の入力と１つの出力のみを有し、２つ以上の回路から出力されるパケットを１つの出力に合流させる処理を行うようになっている。

【００１５】

本発明のある重要な局面は、図２（ａ）に示されるように、入力デバイスあるいは出力デバイスと回路ブロックの間、または回路ブロックの間での信号の伝達を、データセット単位で行うことにある。そして、回路ブロックには、動作に必要な値を含むパケットが入力され、回路ブロック内で所定の処理を行い、結果の値を含むパケットが出力される。回路ブロック間相互の情報伝達がパケットによって行われるため、回路ブロック設計者は複雑な入出力のタイミングの制御という課題から開放される。さらに、図２（ｂ）に示したように、データセット単位でのパイプライン処理が可能となるで、回路ブロックを含む装置に与えられた複数の処理を効率的に行い、全体的な処理速度の向上を達成することができる。

【００１６】

本発明で用いられる「回路ブロック」とは、設計者が任意に設定することができる１ないし複数の特定の処理機能を有するハードウェアモジュールである。この回路ブロックは、ソフトウェア設計におけるＣ言語の「関数」と類似している。また、本発明で用いられる回路ブロックには、同期式回路、非同期式回路、順序回路、ハードワイヤード、マイクロプロセッサ方式などの回路が含まれる。

【００１７】

本発明の別の重要な局面は、図３に示した、ＵＰＬ（Universal Protocol Line）の採用にある。ここで、ＵＰＬとは、パケットによって回路ブロック間の情報伝達を行う回路ブロックを含む装置における情報伝達機構の総称である。ＵＰＬにおいては、入力デバイスあるいは出力デバイスと回路ブロック、または回路ブロック間の信号の伝達はパケットによって行われ、入力された情報パケットは回路ブロックにおける処理を経て出力側に情報パケットとして伝達される、という単一の方向性を持っている。

【００１８】

また、ＵＰＬは、回路ブロックの仕組みといった物理的な取り決め（ビット幅、データ速度、データ有効化方式、データ符号化など）を含むＵＰＬインターフェースと、情報の仕組みのような論理的な取り決め（パケットフォーマットなど）を含むＵＰＬパケットの二つの取り決め（規格）から成る。

【００１９】

ＵＰＬを利用する装置は、ＵＰＬの手法により入出力が単純化された、２種類のＵＰＬ回路：「ＵＰＬ処理回路」と「ＵＰＬ合流回路」を含む。図４（ａ）に示したように、これらの２種類の回路の任意の組み合わせにより、所望の機能を実現するハードウェアを容易に構築することができる。

【００２０】

ＵＰＬ処理回路（図４（ｂ））とは、ＵＰＬ規格に従う入出力をもつ回路ブロックの総称で、０または１個の入力ＵＰＬ（０個の入力ＵＰＬ回路とは、例えば発振器など）と０個以上の出力ＵＰＬ（０個の出力ＵＰＬとはＵＰＬ信号の出力がない、例えば表示デバイス）を有する。ＵＰＬ処理回路は入力データから何らかの計算などの処理を施し、（処理の結果を別の回路が必要とする場合）ＵＰＬ出力からパケットとして結果を出力する。ＵＰＬ処理回路には、計算、遅延、記憶、外部Ｉ／Ｏとのインターフェースなどが含まれる。

【００２１】

ＵＰＬ合流回路（図４（ｃ））とは、ＵＰＬ規格に従う入出力インターフェース持つ回路のことであり、２つ以上の入力ＵＰＬと１つの出力ＵＰＬを有する。ＵＰＬ合流回路は、値の変更などの実質的な処理を行わず、２つ以上の回路ブロックから出力されるパケットを、パケット同士が衝突しないようにしながら、１つのＵＰＬに合流させる機能しか持たない。［０００７］で述べたように、複数の入力を任意のタイミングで受け付ける回路の設計は、容易ではない。しかし、任意の回路を実現しようとすれば、複数の入力を任意のタイミングで受け付ける回路は必ず必要である。本発明では、この設計が容易ではない複数入力を任意のタイミングで受け付ける機能を、ＵＰＬ合流回路に集約した。装置の設計者は、ＵＰＬ合流回路に注力して注意深く設計し、実装、デバッグすればよい。また、一般的にはＵＰＬ合流回路は一度だけ実装すれば、ライブラリとして他の装置への流用も可能である。この結果、装置の設計者は、入力を１個または０個しか持たない比較的設計が容易なＵＰＬ処理回路の設計、実装、デバッグに集中でき、装置の開発効率や品質を飛躍的に高めることが可能となる。

【００２２】

本発明のＵＰＬ装置は従来のハードウェア専用計算機に比べて容易に設計する

ことができる。本発明のＵＰＬ装置に含まれる回路ブロックは前述したように、一ないし複数の機能を有し、これはＣ言語のようなプログラミング言語の関数に類似している。そして、Ｃ言語のようなプログラミング言語で作成したプログラムを適当なトランスレータ（コンパイラ）を用いて主にゲートアレイを基礎とした回路の設計図に変換することが可能であり、これによって回路設計に関する特殊な知識を必要とすることなく容易に本発明のＵＰＬ装置を設計することができる。

【００２３】

さらに本件発明のUPL装置では、装置のハードウェア部分もソフトウェア部分もすべて、まずソフトウェアプログラムで記述されるので、コンピューター上でハードウェア部分を製作することなしにその動作検証を行うことができる。また、ハードウェア開発後のデバッグにはハードウェア部分の基となったソフトウェアプログラムを利用してデバックを行うことができる。従ってデバッグ作業を短時間で高効率に行うことができる。

【００２４】

【発明の実施の形態】

以下、本発明の実施形態を図面を参照して詳細に説明する。本発明の実施形態を、主として、インターネットサーバ機能を実現するＵＰＬ装置に基づいて説明するが、本発明がインターネットサーバに限られず、データマイニング、自然言語処理、ネットワーク情報処理、ＤＮＡ計算シミュレータ、物理シミュレーション、および音声・画像処理のような様々な情報の伝達および計算処理に利用できるであろうことは、当業者には容易に想到し得ることである。

【００２５】

Ｉ．ハードウェア化の対象となるソフトウェア

はじめに、本発明によるハードウェア化の対象となるソフトウェアについて説明する。

【００２６】

今日開発されている情報の伝達および処理用のソフトウェアの規模は非常に大きなものであり、一人のプログラマーが単独で開発を完了することは極めて困難

である。そのため、大規模のソフトウェアをいくつかの機能部分に分割し、その各機能部分を複数人のプログラマーの各人に割り当て、個々に独立してソフトウェア開発を行うようにしている。そして、ソフトウェア開発後に、それぞれ独立に開発された個々のソフトウェアを結合して、ソフトウェア全体を完成させるようになっている。こうすることにより、プログラマーは個別機能の開発に力を集中することができ、ひいては最終的な製品の完成度を高めることにつながる。

【００２７】

ここで、ソフトウェアを所定の機能を有する複数の部分（例えば、関数）に分解し、独立して開発するにあたって、ソフトウェアをどのような部分に分解するか、また部分どうしをどのように結合するかが課題となる。

これに関しては、プログラミング言語に応じて種々の手法が提案されている。

このうちで、本発明においては、「Ｃ言語等の関数呼び出し」の手法を採用して、ソフトウェアの一部もしくは全体をハードウェア化することとしている。

【００２８】

次に、本発明によりハードウェア化するソフトウェアの形態について説明する。ソフトウェアの一つの機能を実現する部分をモジュールと呼ぶことにする。ソフトウェアは一つ以上のモジュールから構成され、モジュールの間は関数呼び出しによって実現されているものとする。以下、プログラミング言語として、Ｃ言語を例にとる。このとき、図５に示されるように、モジュールＡからモジュールＢの関数ｆｕｎｃを呼び出す場合を考える。

【００２９】

通常、ある関数を呼び出すときには引数が指定される。呼び出された関数では、この引数に応じて処理を行う。関数ｆｕｎｃの引数として、整数や浮動小数点などの定数、メモリ上の場所（アドレス）を指示するポインタがある。通常はこれらの値を引数リストとして記述する。これを図６に示されるように、一つのポインタ変数を引数とする関数呼び出しを行うように書き換えることも可能である。

【００３０】

具体的には、関数の引数をすべて一つの構成要素にまとめる。そして、整数や

浮動小数点の値は、その値を構造体の構成要素とする。ポインタ変数の場合は、そのポインタ変数の指示する場所の値を、構造体の構成要素とする。

【００３１】

呼び出された関数側では、この構造体へのポインタが構造体の引数として渡されるので、構造体から引数を再構築し、本来の処理を行うこととなる。

【００３２】

このように、ソフトウェアのプログラミング言語では、もともとの引数を構造体にまとめ、構造体のポインタだけを引数として呼び出しすることでモジュール間の情報交換を行っている。本発明では、この引数の構造体のようなデータのパケットを用いてハードウェア化されたソフトウェアプログラムのモジュール間の情報交換を行う。

【００３３】

次に、モジュールの関数が複数ある場合を考える。図７に示されるようにモジュールＢにはｆｕｎｃ１とｆｕｎｃ２の２つがあるとする。ハードウェア化の観点から鑑みるとハードウェアインターフェースを１つに統一・共通化したほうが、回路規模を小さくできる場合がある。（このことは消費電力の低減やトランジスタの節約の観点からも重要であるといってよかろう。）このためには、モジュールＡから呼び出される関数に対応するものを１つにすればよい。つまり、関数の引数を一つに統一・共通化する実現方法として前述の構造体の構成要素として、本来呼び出したい関数の番号を内包する。これにより、関数によって引数の構造体の定義が違っていたとしても、この番号を見ることによってどの構造体の定義が用いられているかを判別することができる。逆にいえば、どのような種類の構造体が用いられており、その構造体を入力としてどの関数で処理を行えばよいかを関数番号があらわしていると考えることもできる。

【００３４】

つまり、図８に示されるように、モジュールＢでは、モジュールＡから呼び出される関数を、関数番号と構造体をポインタとする統一関数ｆｕｎｃで規定する。この統一関数ｆｕｎｃでは、関数番号ｆｕｎｃＩＤによってｆｕｎｃ１やｆｕｎｃ２を呼び出すことが可能である。

【００３５】

以上、Ｃ言語を例にして説明してきたが、勿論、Ｃ言語以外の言語においても、上述した本発明の手法が適用できる。というのは、コンピュータの基礎理論である情報数学において、通常利用されているアセンブラ言語・ＢＡＳＩＣ言語・Ｃ言語・Ｃ＋＋言語・Ｊａｖａ言語等は各言語の記述能力が等価であることが証明されているからである。このことは、どの言語で書かれているソフトウェアでも他の言語に変換することが可能であることを意味し、本発明の手法を他の言語に適用することを妨げるものではない。

【００３６】

II．ハードウェア化の基本手法

次に、本発明によるソフトウェアをハードウェア化するための基本的手法について説明する。本発明における、ハードウェア化されるソフトウェアのモジュールは、図８に示す形を有している必要がある。ここでは、モジュール間でどのように引数構造体が扱われているか説明する。図９にその代表例を示す。モジュールＢを見ると外部（この例ではモジュールＡ）から引数argsに対応するメモリ領域が示され、この領域の情報を使って何らかの計算・処理を行う。モジュールＢもモジュールＣを呼び出すことが一般的であるから、モジュールＣに対して、引数構造体のポインタを指定して関数を呼び出すことになる。つまり、モジュールの入力として引数構造体が渡され、モジュールはその構造体に書かれた情報から計算を行い、次のモジュールを呼び出す引数構造体を生成する。すべてのソフトウェアは、このような形式で実現することができる。

【００３７】

本発明においてプログラミング言語で記述されたソフトウェアモジュールをハードウェア化する場合は、まず、ｆｕｎｃ１やｆｕｎｃ２の処理本体の関数部分をハードウェアで実現する。また引数構造体は、レジスタとして関数部分のハードウェアから参照することができる。

【００３８】

つまり、図１０に示されるように、入力としての引数構造体に対応する入力レジスタと次のモジュール呼び出しのための引数構造体に対する出力レジスタが存

在する。入力レジスタの値から、ハードウェアによって計算を行い、結果を出力レジスタに格納する。ハードウェアとしては、ハードワイヤード方式の回路を用いることも状態遷移機械を用いることも可能である。状態遷移機械の遷移や出力がプログラム可能なものとして、マイクロプロセッサがあるが、マイクロプロセッサによって入力レジスタから出力レジスタの値を計算することも可能である。

【００３９】

III. ハードウェアモジュール間の通信方法

さらに、本発明によるハードウェアモジュール間の通信方法特に、本発明において重要なモジュール間通信システムの仕組みについて説明する。モジュール間の関数呼び出しにおける引数には、関数番号と引数構造体のポインタがあることを前節において述べたとおりである。計算に係るハードウェアから見ると、引数構造体の実体がレジスタの出力値として参照できる必要があることから、実際はポインタではなく実体である値が必要である。つまり、モジュールをなすハードウェア間で、関数番号と引数構造体の実体である値をやり取りすればよい。

【００４０】

図１１にモジュール間でやりとりするパケットの例を示す。関数番号と引数構造体からなるデータをパケットというデータの塊にまとめ、このパケット単位でモジュール間通信を行う。パケットの先頭１ワードには、関数番号が格納されており、この番号をもとにしてどの計算ハードウェアの入力レジスタに引数構造体を反映させるかを決定する。図１２には、パケットの入出力ハードウェアの例を示す。モジュールハードウェア間には、ユニバーサルプロトコルライン（ＵＰＬ）と便宜上呼ぶパケット通信を採用し、これによってモジュール間が接続される。ＵＰＬは出力レジスタから入力レジスタへとその値を転送する仕組みである。ＵＰＬの実装には、通常使われるシリアル転送の仕組みやＬＶＤＳ（Low Voltage Differential Signaling）、３値論理方式、Ethernet、USB、IEEE1284、インターネットなど、転送速度、伝送距離などの諸条件にあったものを自由に選ぶことができる。

【００４１】

出力レジスタは関数番号と引数構造体に相当するデータ部からなる。モジュー

ルハードウェアによって、出力レジスタに値が設定されると、出力レジスタの値がＵＰＬの出力される。ＵＰＬによって受信されるパケットのうち、当該受信モジュールが受信すべきパケットだけを入力レジスタに設定する。受信すべきかどうかは、関数番号を参照することで判断できる。

【００４２】

IV. ＵＰＬ処理回路およびＵＰＬ合流回路の具体例

図１３に1つのＵＰＬ処理回路の具体例を示した。ＵＰＬパケットデータは、入力ＵＰＬとしてデータ信号線を介してＵＰＬ処理回路部に入力され、ＵＰＬ入力回路部において個々の処理を行う演算回路に適したデータ形式に変換されてユーザ処理回路に出力され、ユーザ処理回路部で所定の演算処理などの所定の処理を加えてＵＰＬ出力回路に出力され、ＵＰＬ出力回路部ではユーザ処理回路からの出力をＵＰＬ仕様に基づくＵＰＬパケットデータに変換されて出力ＵＰＬとして出力される。このようにＵＰＬ回路間でＵＰＬパケットデータが伝送される。また、送受されるデータはイネーブル信号およびクロックによってタイミングなどの制御が行われている。

【００４３】

より具体的には、ＵＰＬ入力回路に入力されたＵＰＬデータパケットはシリアルインパラレルアウトレジスタにおいて、それぞれの回路に適したデータの入力形式に分割される。図において太線は８ビットに相当する複線の信号線を意味している。つづいてｆｕｎｃＩＤとして収容されたデータが処理ステートマシンによって評価され、入力されたデータが自回路あてのデータである場合、シリアルインパラレルアウトレジスタに収容された入力値ａ，ｂを加算器のＡ、Ｂから入力して加算し、結果ｅを出力して比較器のＣへ入力する。また、比較器のＤへの入力値ｃが入力されており、比較器においてｅとｃの大きさが比較される。ここでｅ＞ｃであれば、比較器からｆ＝１が、ｅ＜ｃであればｆ＝０が入力値ｆとしてマルチプレクサに出力される。つぎにマルチプレクサにおいて、ｆの値に基づいてｆ＝１ならば、Ｅからの入力値ｃを、ｆ＝０ならば、Ｆからの入力値ｄをＵＰＬ出力回路に入力値ｇとして出力する。最後に処理ステートマシンから出力された次に利用される回路を示すデータとマルチプレクサから出力されたｇをＵＰ

Ｌ出力回路部においてＵＰＬパケットデータに変換して出力する。処理ステートマシンによって評価されたデータが自回路あてではない場合は、このＵＰＬ処理回路は特定の処理を行わず、したがって、データは出力されない。ここでは図示されていないが、自回路あてではないデータを入力された場合にそのデータをそのまま次の回路へ出力する経路を持つことも可能である。この例では、入力ＵＰＬのデータ線の幅は１bitである。データ線の幅は、２bitでも３bitでもデータの全長である４０bitでも、それより長い例えば１２８bitでもよい。データ線の幅を広くすれば、データの伝送にかかる時間を短くすることができるが、ＵＰＬ回路間の配線数が増える。設計者は、回路に要求された仕様や、回路実現に用いるＬＳＩなどの素子の特性、物理的な配線の制約などを考慮して、最適なデータ幅を用いることができる。

【００４４】

以下にＵＰＬ処理回路の各部分のより詳しい説明を行う。

入力ＵＰＬ

この例では、１ビットデータ線からなるクロック同期シリアル伝送路である。データ線上の値がデータとして有効であることを示すイネーブル信号線を伴っている。

ＵＰＬ入力回路

ＵＰＬ信号線群をユーザ処理回路の入力値として接続する回路。

シリアルインパラレルアウトレジスタ

クロック同期型のシフトレジスタにより構成される。イネーブル信号線が有効であるとき、入力された１ビットデータを順にＱ０からＱ３９にセットする。

入力ステートマシン

入力ＵＰＬのイネーブル信号を受け取り、シリアルインパラレルアウトレジスタを制御する。またパケットの受信が完了したことを検出する。受信が完了するとｆｕｎｃＩＤ，ａ，ｂ，ｃ，ｄの各値が確定するので、入力イネーブル信号を出力し、次の処理ステートマシンにそのことを通知する。

処理ステートマシン

入力値が確定した後、出力値が確定するタイミングを生成する回路。ユーザ処

理回路中のレジスタの確定タイミングも制御する。

ｆｕｎｃＩＤを入力として受け取り、データが自回路宛てであるかどうかを判断する。自回路宛ての場合は、処理回路の動作を制御し、出力を行う。また、自回路宛てでない場合は、処理回路の動作を停止し、出力は行わない。

ｆｕｎｃＩＤを出力し、次に動作を行う回路の指定を行う。場合によってはユーザ処理回路からの信号により、ｆｕｎｃＩＤを変化させ、処理結果に応じて次に動作を行う回路を動的に変化させる。

入力イネーブル信号

ユーザ処理回路の入力値が確定したことを示す信号線。

出力イネーブル信号

ユーザ処理回路の出力値が確定したことを示す信号線。

ユーザ処理回路

ＵＰＬ処理回路が本来行いたい処理そのものを実現する回路。通常はＵＰＬ装置設計者がC言語等のプログラミング言語で記述したプログラムから、コンパイラによって自動合成される。設計者がVHDLやVerilogなどの回路記述言語により直接記述することもある。

出力ＵＰＬ

この例では、１ビットデータ線からなるクロック同期シリアル伝送路である。データ線上の値がデータとして有効であることを示すイネーブル信号線を伴っている。

ＵＰＬ出力回路

ユーザ処理回路の出力値を出力ＵＰＬ信号線群に接続する回路。

パラレルインシリアルアウトレジスタ

クロック同期型のシフトレジスタにより構成される。イネーブル信号線が有効であるとき、入力されたＤ０からＤ１５の値を内部ラッチに保持する。
その後、クロック毎にＤ０から１ビットずつＱに出力される。

出力ステートマシン

出力イネーブル信号を受け取り、パケットの送信を制御する。パラレルインシリアルアウトレジスタを制御し、出力ＵＰＬのイネーブル信号を生成する。

【００４５】

図１４に1つのＵＰＬ合流回路の具体例を示した。ＵＰＬパケットデータは、入力ＵＰＬ１および入力ＵＰＬ２としてデータ信号線を介して、入力ＵＰＬ１と入力ＵＰＬ２のタイミングをクロック調整回路によって調整しながら、ＵＰＬ合流回路に入力される。それぞれの入力ＵＰＬから入力されたデータはそれぞれのパケットバッファメモリに保持される。パケットバッファメモリはそれぞれのパケットバッファメモリに接続されたパケットバッファメモリ管理ステートマシンによって、その書き込み／読み込みの制御が行われる。また、パケットバッファメモリ管理ステートマシンはさらに出力調停回路に接続されている。そして入力ＵＰＬ１および入力ＵＰＬ２から入力されたＵＰＬパケットデータは、出力調停回路によってタイミングの調整をされながら、単一の出力ＵＰＬからＵＰＬパケットデータとして出力される。また、送受されるデータはイネーブル信号およびクロックによってタイミングなどの制御がされている。

【００４６】

以下にＵＰＬ合流回路の各部分のより詳しい説明を行う。

入力ＵＰＬ１、入力ＵＰＬ２

この例では、１ビットデータ線からなるクロック同期シリアル伝送路である。データ線上の値がデータとして有効であることを示すイネーブル信号線を伴っている。

パケットバッファメモリ

入力ＵＰＬから入力されたパケットを一時的に保持するメモリ。

パケットバッファメモリ管理ステートマシン

パケットバッファメモリの書き込み、読み込みの制御を行い、出力ＵＰＬのイネーブル信号線を制御する。

複数のパケットバッファメモリが同時に出力を行うと、パケットのデータが破壊されてしまうため、排他的に処理を行うための仕組みが必要である。これを実現するために出力調停回路に調停を要求し、許可があったときのみ出力ＵＰＬに出力する処理を行う。

出力調停回路

調停要求入力を複数回路から受け取り、要求を出した回路に対して、調停確認出力を返す。同時に調停要求入力を受け取ったときは、同時には１つの調停確認出力だけを返す。

出力ＵＰＬ

ＵＰＬ合流回路の出力ＵＰＬであり、複数の入力ＵＰＬから受け取ったパケットを値を変えずに出力する。

【００４７】

Ｖ．本発明の応用例

上述したような本発明の技術手法を用いることによって，これまではソフトウェアで実現されていた様々なデータ処理装置を容易にハードウェア化することが可能となる。以下でその具体的な応用例について説明する。

【００４８】

インターネットサーバ

インターネット上でサービスを提供するサーバといわれるコンピュータがある。インターネットでつながったクライアントコンピュータから様々な要求を受理し、その要求に応じてクライアントにデータを返すコンピュータである。このコンピュータには、オペレーティングシステムソフトウェアとサーバソフトウェアが動作している。インターネットが爆発的に拡大し、クライアントコンピュータ数が増加し、またアクセス回線がＡＤＳＬ・ＳＤＳＬ・光ファイバー等により、乗数倍的に高速化されているため、大量の要求がサーバコンピュータに集中的に負荷がかかっている。

【００４９】

これまでは、サーバコンピュータのＣＰＵやメモリといったハードウェア的な処理能力を増強し、また、ソフトウェアの改良によって、処理可能な量を等差的に増大させてきた。しかしながら、もはや要求の増加のほうが処理能力を上回りつつあるし、現在すでに一時的に処理能力を上回ったため、サーバが機能できなくなり、サービスを停止してしまうということがしばしばみうけられる。

【００５０】

そこで、本発明の技術手法を情報処理装置に適用し、サーバ上で動作している

オペレーティングソフトウェアとサーバソフトウェアをハードウェアに置き換えるようにすれば、処理を高速に行うことが可能となる。この他にもルータ、クライアントマシンにおいても同様のことがあてはまる。

【００５１】

サーバソフトウェアの例として、Ｗｅｂサーバのハードウェア化の例を図１５に示す。イーサネット受信モジュールによって、ＬＡＮの一種であるイーサネット物理層インタフェースが受信されたイーサパケットを処理し、ＵＰＬに出力する。ＵＰＬ上には、ＡＲＰ（アドレス解決プロトコル）を処理するＡＲＰモジュール、ＩＰ（インターネットプロトコル）を処理するＩＰモジュールがつながっている。イーサネットパケットのプロトコル識別子の値に応じて別々のプロトコルコードをつけられたパケットが生成される。これによって、イーサネットモジュールがＵＰＬに送信したパケットは、ＡＲＰモジュールが処理すべきかＩＰモジュールが処理すべきかが決定される。

【００５２】

イーサネットモジュールのＡＲＰ回路は、ＡＲＰリクエストパケットを受信したときに、ＡＲＰリプライパケットを送信するという処理を担当するモジュールである。データ部に含まれるイーサネットパケットを参照しながら、リプライパケットを送信すべきかどうか、どういう内容のリプライパケットかを、自ＩＰアドレスなどを保持している情報テーブルを参照しながら決定し、イーサネット送信モジュールに向けて送信する。

【００５３】

ＩＰモジュールのＩＰ受信回路では、ＩＰパケットを受信し、チェックサムの検査、受信すべきＩＰパケットかどうかの検査を行う。さらにＩＰデータ部のプロトコルに応じて、ＴＣＰパケットであればＴＣＰモジュールへ、ＵＤＰパケットであればＵＤＰモジュールへ、といった分岐処理を行う。このとき、ＩＰヘッダの中から、上位層の処理で必要なＩＰアドレスやパケット長などの情報を抽出し、ＩＰ層不完全ヘッダとして、ＩＰ層データとともに、送信される。また、分断化されたパケットの復元などの処理、自分宛でないパケットのフォワード処理なども適宜行う。またＴＣＰモジュールやＩＣＭＰ回路、その他の回路からのＵ

ＰＬをＵＰＬ合流回路により１つに集約し、ＩＰモジュールのＩＰ送信回路に接続される。送信先に関する情報がＩＰ層不完全ヘッダとして、ＴＣＰなど前段モジュールから与えられる。これらの情報から、ＩＰ送信回路ではＩＰ層完全ヘッダを構成し、イーサネット送信回路へ出力する。ＴＣＰモジュールでは、ＴＣＰで定めたプロトコル処理を、また、ＨＴＴＰモジュールでは、ＨＴＴＰで定められたプロトコル処理を行う。

【００５４】

コンテンツモジュールでは、Ｗｅｂサーバがクライアントコンピュータに送り返すべきＷｅｂデータを保持している。保持機構としては、フラッシュメモリ・スタティックメモリ等電気的記憶装置、このほかにもハードディスクといった磁気的記憶装置等、電気回路と接続可能な様々な記憶装置をもちいることができる。

【００５５】

ＯＳＩ７階層モデル

図１６（ａ）に、ネットワークの基本となる構想であるＯＳＩ７階層モデルに本発明のＵＰＬを適用した例を示す。ＯＳＩ７階層モデルは、図のように第１層（物理層）、第２層（データリンク層）、第３層（ネットワーク層）、第４層（トランスポート層）、第５層（セッション層）、第６層（プレゼンテーション層）、第７層（アプリケーション層）を含んでいる。ここで各層は、第Ｎ層送信回路７と第Ｎ層受信回路６によって接続され、それぞれの層がＵＰＬ規格に従った伝送方式によって隣り合った層との間で通信を行っている。また、第１層の処理回路に接続された外部受信回路、外部送信回路、コネクタおよびケーブルを介して外部のネットワークと接続されている。外部のネットワークとの通信には当業者には周知の転送方式を使用することもできる。図１６（ｂ）は、ＯＳＩ７階層モデルの隣り合う層の接続部分の拡大図を示す。ここでは、Ｎ層内のモジュールＡの処理装置のＵＰＬ出力からＵＰＬパケットが出力され、入力レジスタ、通信路、出力レジスタを介して隣接するＭ層(M=N+1 or N-1)内のモジュールＢの処理装置のＵＰＬ入力にＵＰＬパケットが伝送されるようになっている。

【００５６】

ここで用いられるデータパケットは、例えば、図１７（ａ）のような構造を有している。図１７（ｂ）には第Ｎ層受信回路６を示した。ここで、Ｎ－１層までの完全ヘッダのうち、Ｎ層の処理で必要な情報を抽出したものを「Ｎ－１層の不完全ヘッダ情報」と呼ぶことにする。(完全ヘッダとは、通信プロトコルでの取り決めに従って構成されているヘッダ情報である。)一般に「Ｎ－１層のデータ」は、Ｎ層において「Ｎ層の完全ヘッダ」と「Ｎ層のデータ」ととらえることができる。このときＮ層では、「Ｎ－１層の不完全ヘッダ情報」と「Ｎ層の完全ヘッダ情報」からＮ＋１層の処理で必要となる「Ｎ層の不完全ヘッダ情報」を生成する。そして生成された「Ｎ層の不完全ヘッダ情報」と「Ｎ層のデータ」は、UPLを介してＮ＋１層に伝達される。また、送信処理においては上記とまったく逆の動作が行なわれる。図１７（ｃ）に第Ｎ層送信回路７を示した。Ｎ＋１層から、「Ｎ層の不完全ヘッダ情報」と「Ｎ層のデータ」がＵＰＬを介してＮ層に伝達されてくる。Ｎ層の処理回路では、「Ｎ層の不完全ヘッダ情報」にＮ層内部に持っている情報を追加して、「Ｎ層の完全ヘッダ情報」を生成する。そして、「Ｎ層の完全ヘッダ情報」と「Ｎ層のデータ」を合わせて「Ｎ－１層のデータ」とする。また「Ｎ－１層の不完全ヘッダ情報」も生成する。そして「Ｎ－１層の不完全ヘッダ情報」と「Ｎ－１層のデータ」をＵＰＬを介してＮ－１層に伝送する。

【００５７】

このように、本発明ではインターネット環境における情報通信処理装置において、これまでの一貫したＣＰＵ処理に代わる、ヘッダの完全化・不完全化による通信処理装置とＵＰＬ装置の採用によって、通信情報処理装置の通信処理ソフトウェアを効果的にハードウェアに置き換えることが可能となり、通信プロトコルの高速処理が実現される。この結果、ワイヤースピードで通信処理をすることが可能となる。

【００５８】

ここで、ヘッダの不完全化による通信情報処理とは、外部入力からの情報を後続の回路での処理において必要なものだけに共通化・抽象化することによって処理を迅速にする手法である。この点、もともとのデータ量と比較すると情報量は

減らされているが、内部処理において情報流通に必要なものだけを故意に共通化・抽象化し、不完全化することにより、プロトコル処理を高速化することが可能となる。この手法を使って、情報処理装置におけるプロトコル処理を円滑に処理することができる。

【００５９】

上述のように、本発明により、ＵＰＬ装置を用いてＯＳＩ７階層モデルにおける各層の入出力処理をハードウェアで実現することができるが、出力に関して、上記ＵＰＬ装置とヘッダの完全化による情報処理装置を用いて、ハードウェア上で不完全ヘッダ情報を完全ヘッダ情報に戻し、必要に応じて情報テーブルを参照しながら、必要な情報をヘッダと一緒に要求された装置に迅速に送ることができるようになっている。

【００６０】

その他の応用例

本発明の他の応用例（ａ）によれば、図１８に示すように、回路間の通信量や、ＬＳＩ間の物理的制約にあわせて、任意に選択された１ないし複数のＵＰＬ処理回路（およびＵＰＬ合流回路）を１つのＬＳＩに割付して、複数のＬＳＩの組み合わせからＵＰＬ装置を構成することができる。ここで、ＬＳＩ１を例にとると、複数の入力（図では２つ）をもつＵＰＬ合流回路が、２つ以上の回路から出力されるパケットを１つのパケットに単純に合流させて出力する処理を行う。そして、つづくＵＰＬ処理回路が所定の処理（計算など）を行い結果をパケットとして出力し、つづくＬＳＩ２にパケットを伝送している。また、ここではＵＰＬ処理回路の出力の１つをＵＰＬ合流回路にフィードバックさせている。

【００６１】

本発明の他実施例（ｂ）によれば、図１９（ａ）のような回路においてＵＰＬ処理回路の処理速度がボトルネックとなっている回路を、図１９（ｂ）に示すように単体の処理速度を高速化する、または図１９（ｃ）に示すようにＵＰＬ処理回路を並列に設置することによって容易にボトルネックとなっているＵＰＬ処理回路の処理スピードを改善し、他の回路の実装を変更せずとも、装置全体の処理能力を向上することができる。

【００６２】

本発明の他の応用例（ｃ）によるメモリアクセス回路を図２０示す。この図に示されるように、本発明によるＵＰＬ処理回路を応用したメモリアクセス回路は、メモリのリード／ライトをするＵＰＬ処理回路と計算等をするＵＰＬ処理回路を分割することによって、図２０（ｂ）に示すようにパイプライン処理を行うことが可能となり、従来より高速な処理ができる（従来はメモリ読み込み、計算処理、メモリ書き込みの一連の動作を完了してから次の処理へと進んでいた）。また、実際の回路では図２０（ｃ）に示すように１つの処理回路で排他処理などのメモリ管理を行うこともできる。

【００６３】

本発明の他の応用例（ｄ）を図２１に示す。この図に示されるように、既存のデータフォーマット（イーサネットパケットなど）の外部情報を、ＵＰＬ処理回路においてＵＰＬパケットに変換しカプセル化することができる。これによって、ＵＰＬの枠組みの中で、一見ＵＰＬとは異なったデータフォーマットも扱うことができ、また既存のデータフォーマットを処理する回路を構成することもできる。

【００６４】

本発明の他の応用例（ｅ）を図２２に示す。この図に示されるように、本発明のＵＰＬ装置はプログラミング言語との高い親和性を持つので、Ｃ言語、Ｃ＋＋言語、Ｊａｖａ言語などの関数（メソッド）をＵＰＬ処理回路に対応付けすることができる。また、ＵＰＬ処理回路単位でのデバッグ作業が容易にできる。

【００６５】

本発明の他の応用例（ｆ）を図２３に示す。前述したように、高効率メモリアクセスの実現と、メソッドのハードウェア化の組み合わせによって、オブジェクト指向ソフトウェアを容易にハードウェアとして実現することができる。図２３（ａ）に示す例においては、オブジェクト指向ソフトウェアをＣ言語に変換し、さらにハードウェアの回路構成に変換することを意図しているが、オブジェクト指向ソフトウェアから直接ハードウェアの回路構成に変換することも可能である。図２３（ｂ）は、具体的なＣ言語プログラムのハードウェア化を示す。この図

に示されるように、クラス変数やインスタンス変数は前述したＵＰＬメモリアクセス回路のメモリ素子に保持し、関数は前述したようにＵＰＬ処理回路に対応付けてハードウェア化することができる。

【００６６】

今までの例においては、ソフトウェアプログラムをすべてハードウェア化する方法を述べてきたが、ハードウェア化が、スピードやコストなど他の点で利点がない場合には一部の機能単位はハードウェア化せずに汎用の計算機で行い、その計算機にＵＰＬ入／出力装置を設けて、ＵＰＬ仕様に基づくハードウェア部分と、汎用計算機によるソフトウェア部分を接続して共存させることも可能である。

【００６７】

以下に、ＵＰＬ装置によるハードウェアとソフトウェアの協調開発の概念を図２４を参照しながら説明する。まず、図２４（ａ）の初期段階において、装置のｎ個の全ての機能（ソフトウェア機能１、ソフトウェア機能２、ソフトウェア機能３、、、ソフトウェア機能ｎ）をソフトウェアで記述したプログラムを構成する。ここでソフトウェアプログラムの構成方法（アルゴリズムやデータ構造、機能分割など）に誤りがないことを確認する。このソフトウェアプログラムの構成は通常のマイクロコンピューターでも行うことができる。

【００６８】

次に図２４（ｂ）のハードウェア段階において、装置としての動作を検証しながら、可能なところからソフトウェアによって構成されていた機能の一部を既に述べたようなＵＰＬ仕様に基づくハードウェアであるＵＰＬ処理回路（およびＵＰＬ合流回路）に順次置き換える。（図ではソフトウェア機能1、ソフトウェア機能２、ソフトウェア機能３がそれぞれＵＰＬ処理回路１、ＵＰＬ処理回路２、ＵＰＬ処理回路３に置き換えられている。また、図示はされていないがソフトウェア部は通常のマイクロコンピューターのような計算機で実行され、ＵＰＬ仕様に基づくハードウェア部分とコンピューター部分を接続するために、その間にコンピューターの入出力をＵＰＬ仕様に変換する装置が存在する。）ここでハードウェア単体の検証と装置としての検証を同時に行うことができる。そして、動作検証を行いながら、ソフトウェアによって実現されていた機能をＵＰＬ処理回路

（およびＵＰＬ合流回路）によりハードウェアに置換する工程をつづける。ＵＰＬ処理回路同士の動作について独立性が高いので、ハードウェア化を行った部分に集中的にデバッグができる。また、ＵＰＬ処理回路同士は別々に開発ができ、個々が完全に動作するとき、それらを接続した全体も完全に動作する。ＵＰＬ処理回路を追加して、動作しなくなれば、その回路が動作しない原因であり、動作しない部分の特定が容易である。さらに、ＵＰＬ処理回路は全体に比べ規模が小さいため、ＵＰＬ処理回路内部のデバッグは容易にできるというような特徴がある。

【００６９】

最後に図２１（ｃ）に開発終了段階を示した。この段階は当初予定したハードウェアとソフトウェアの分割点に達した時点を示す。全ての機能がハードウェア化される装置ではマイクロプロセッサ部分（ソフトウェア部分）がなくなる。また、この段階にいたる全ての時点で常に動作検証が行えるため、この段階に達した時点で、動作検証も完了することができる。

【００７０】

【発明の効果】

本発明によれば、次のような作用効果が得られることとなる。

（１）回路は、ソフトウェアをもたず、ハードワイアードな構成となるため、処理動作を高速化することができる。例えば、前記インターネットサーバに対する応用において、サーバに対して高い負荷がかかっても、情報を処理しきれずに、結果として、処理を停止してしまう事態が回避され、ギガビットレベルで瞬時に処理し、インフラとしてのインターネットに対して信頼をもたせることが可能となる。

（２）回路設計に柔軟性をもたせることができるため、処理能力の変更及び向上が容易である。

（３）回路分割が容易であり、ＡＳＩＣ等の専用ＬＳＩでなく複数の中規模汎用集積回路や大規模ゲートアレイ（例えば、ＦＰＧＡ）により回路が実現できる。

（４）プログラミング言語との親和性が高く、またオブジェクト指向のソフトウェアのハードウェア化が容易である。

（５）ＵＰＬはソフトウェアとハードウェアの統一的なインターフェースであるので、ハードウェアとソフトウェアを協調させることが容易にでき、ハードウェアの斬新的な開発を容易に行うことができ、結果として複雑な装置であっても、短時間に低コストで容易に開発することができる。

【図面の簡単な説明】

【図１】

従来の回路の入出力および、そのタイミング関係を示す図である。

【図２】

本発明のパケットデータに基づく回路の入出力および、そのタイミング関係を示す図である。

【図３】

本発明のＵＰＬの概念を示す図である。

【図４】

本発明による大規模ＵＰＬ回路の構成例を概念的に示す図である。

【図５】

Ｃ言語を用いた関数呼び出しを示す図である。

【図６】

Ｃ言語を用いたモジュール相関を示す図である。

【図７】

Ｃ言語を用いたモジュール相関を示す図である。

【図８】

Ｃ言語を用いたモジュール相関を示す図である。

【図９】

計算（処理）によるパケットの生成過程を示す図である。

【図１０】

パケットの生成に関する詳細な説明図である。

【図１１】

関数番号と引数構造体を示す図である。

【図１２】

ＵＰＬの役割を示す図である。

【図１３】

ＵＰＬ処理回路の具体的な例を示す図である。

【図１４】

ＵＰＬ合流回路の具体的な例を示す図である。

【図１５】

本発明の全体の構成を概念的に示す図である。

【図１６】

ＯＳＩ階層モデルと、これに応用されるＵＰＬの概念図である。

【図１７】

第Ｎ層における受信回路および送信回路を示す図である。

【図１８】

本発明のＵＰＬ大規模回路を複数のＬＳＩに分割した実施例を示す図である。

【図１９】

本発明のＵＰＬ回路の処理速度向上を実現する構成の実施例を示す図である。

【図２０】

本発明によるＵＰＬ回路によるメモリアクセスの高速化を実現する構成の実施例を示す図である。

【図２１】

本発明によるＵＰＬ回路によって外部情報を扱う実施例を示す図である。

【図２２】

本発明のＵＰＬ回路と、ソフトウェアの関数との親和性の高さを表わす図である。

【図２３】

オブジェクト指向ソフトウェアをハード化することを目的とした構成の実施例を示す図である。

【図２４】

ＵＰＬ装置によるハードウェアとソフトウェアの協調開発の概念を示す図である。

【符号の説明】

1　ヘッダの完全化・不完全化による通信処理装置

2　ＵＰＬ（Universal Protocol Line）

3　外部入力からの通信プロトコル情報を共通化・抽象化

4　内部処理における共通化・抽象化した不完全化した通信情報

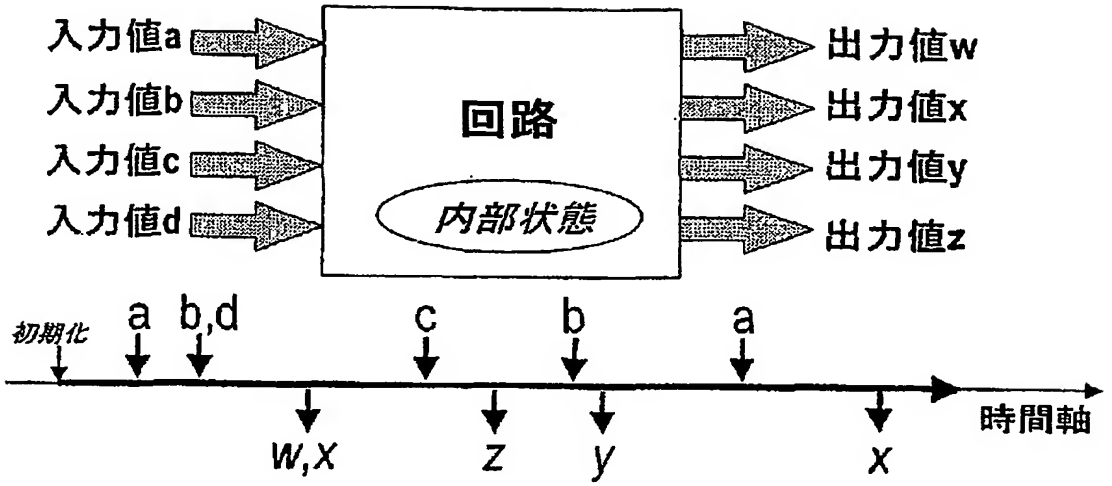
5　ＯＳＩ７階層モデル

6　第Ｎ層受信回路

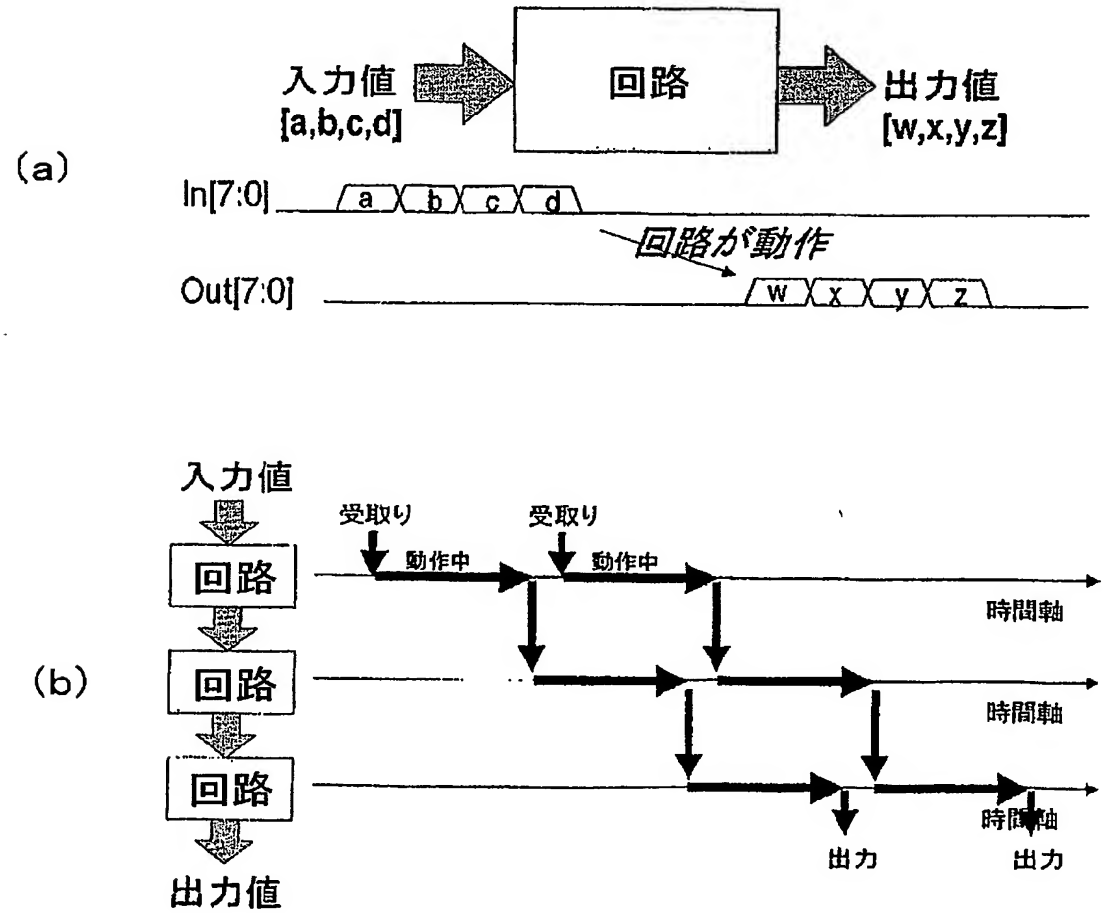
7　第Ｎ層送信回路

【書類名】　図面

【図１】

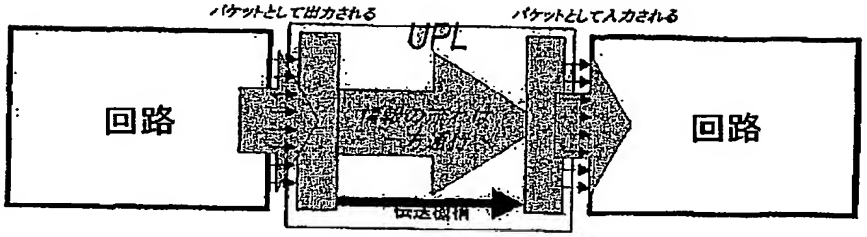

入力値a
入力値b
入力値c
入力値d
回路
内部状態
出力値w
出力値x
出力値y
出力値z
初期化
a
b,d
c
b
a
時間軸
W,X
Z
y
X

【図２】


(a)
入力値
[a,b,c,d]
回路
出力値
[w,x,y,z]
In[7:0]
a
b
c
d
回路が動作
Out[7:0]
w
x
y
z
(b)
入力値
回路
回路
回路
出力値
受取り
動作中
受取り
動作中
時間軸
時間軸
時間軸
出力
出力

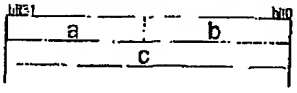
【図３】

(a)



❖UPLインターフェース
- 物理的な取り決め（回路の仕組み）
- ビット幅、データ速度、データ有効化方式、データ符号化など
  - 例: 同期式25MHz,32bit幅、ストローブ信号線によるデータの有効化、データはバイナリエンコーディング

❖UPLパケット
- 論理的な取り決め（情報の仕組み）
- パケットフォーマット
  - パケットのなかでの値の順番付け、意味付け
  - 例:

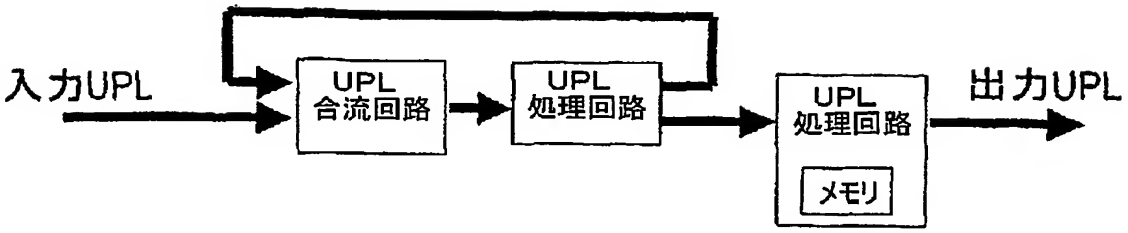


(b)

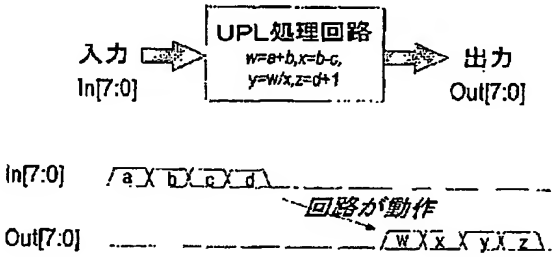
(c)

【図４】

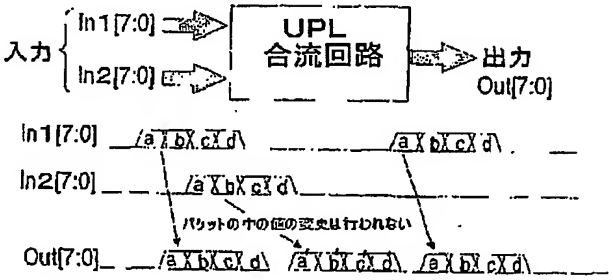
(a)


入力UPL
UPL
合流回路
UPL
処理回路
UPL
処理回路
メモリ
出力UPL

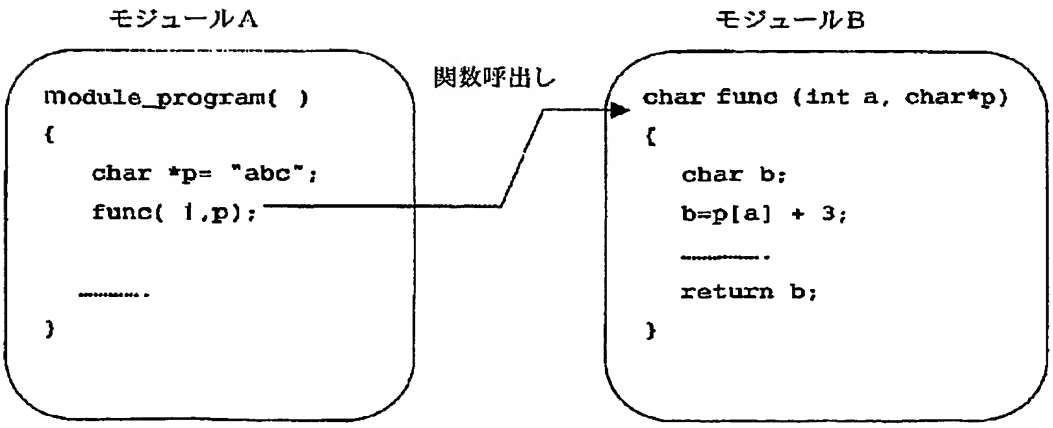
(b)


入力
In[7:0]
UPL処理回路
w=a+b,x=b-c,
y=w/x,z=d+1
出力
Out[7:0]
In[7:0]
a
b
c
d
回路が動作
Out[7:0]
w
x
y
z

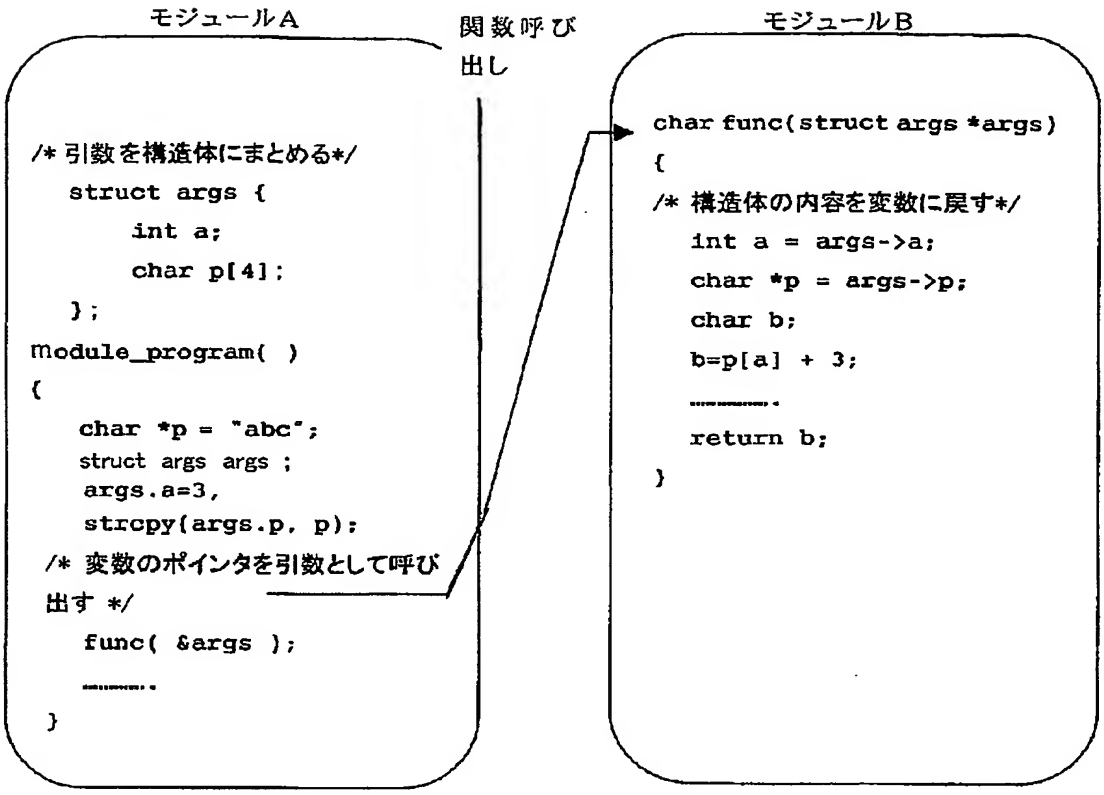
(c)


入力
In1[7:0]
In2[7:0]
UPL
合流回路
出力
Out[7:0]
In1[7:0]
In2[7:0]
パケットの中の値の変更は行われない
Out[7:0]

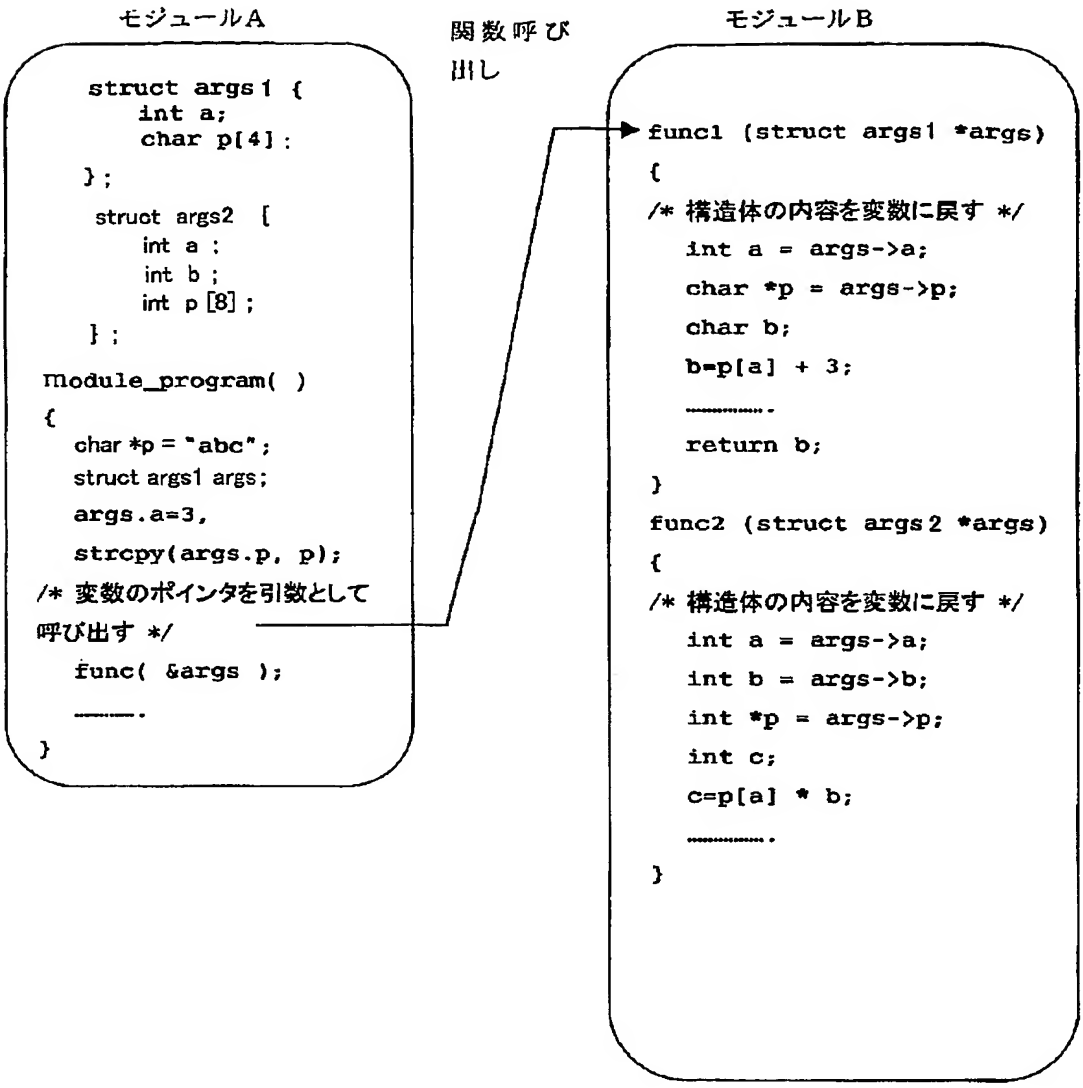
【図５】


モジュールA
module_program( )
{
char *p= "abc";
func(1,p);
.....
}
関数呼出し
モジュールB
char func (int a, char*p)
{
char b;
b=p[a] + 3;
.....
return b;
}

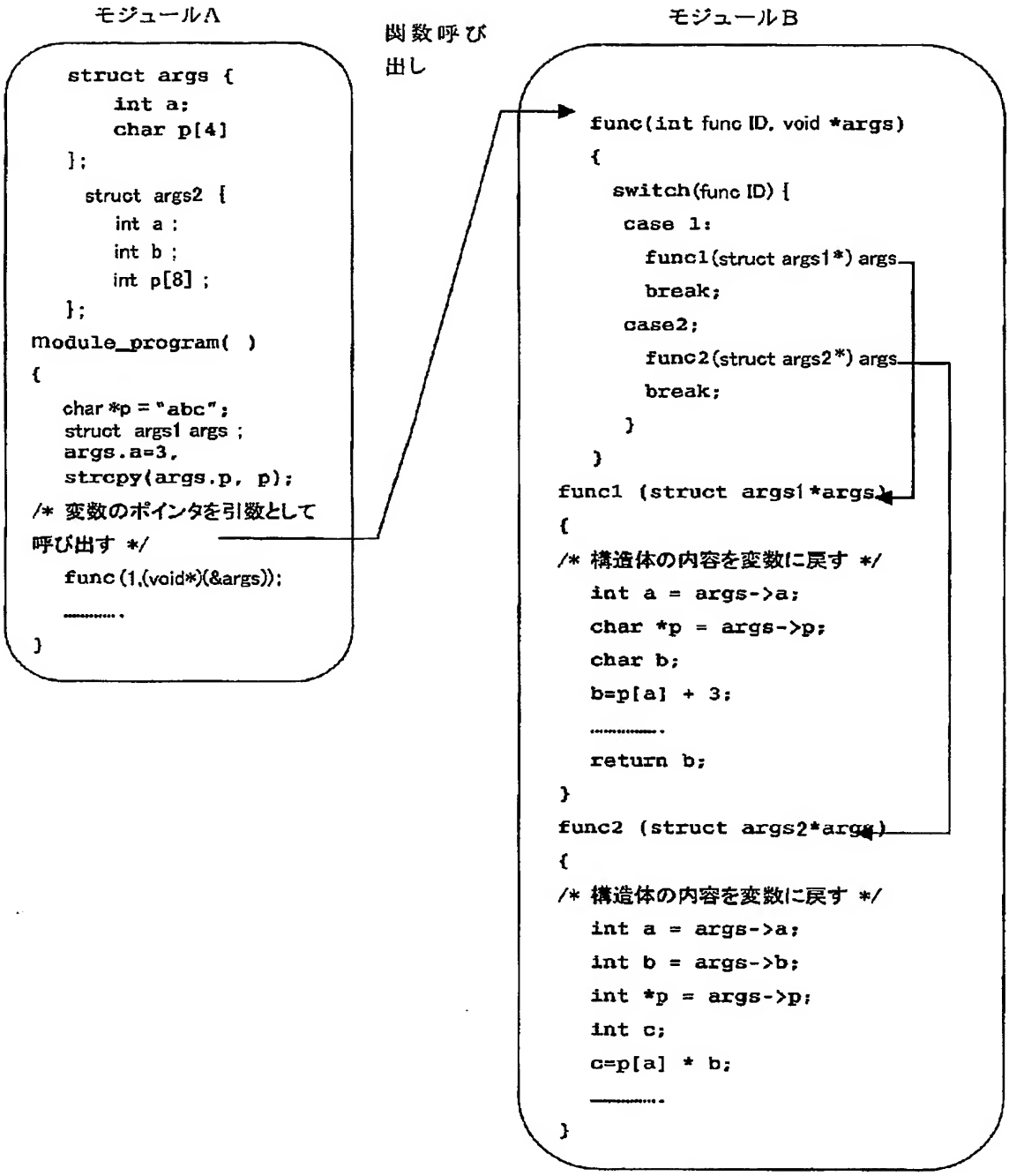
【図６】


モジュールA
関数呼び出し
モジュールB
/* 引数を構造体にまとめる*/
struct args {
int a;
char p[4];
};
module_program( )
{
char *p = "abc";
struct args args ;
args.a=3,
strcpy(args.p, p);
/* 変数のポインタを引数として呼び出す */
func(&args);
.....
}
char func(struct args *args)
{
/* 構造体の内容を変数に戻す*/
int a = args->a;
char *p = args->p;
char b;
b=p[a] + 3;
.....
return b;
}

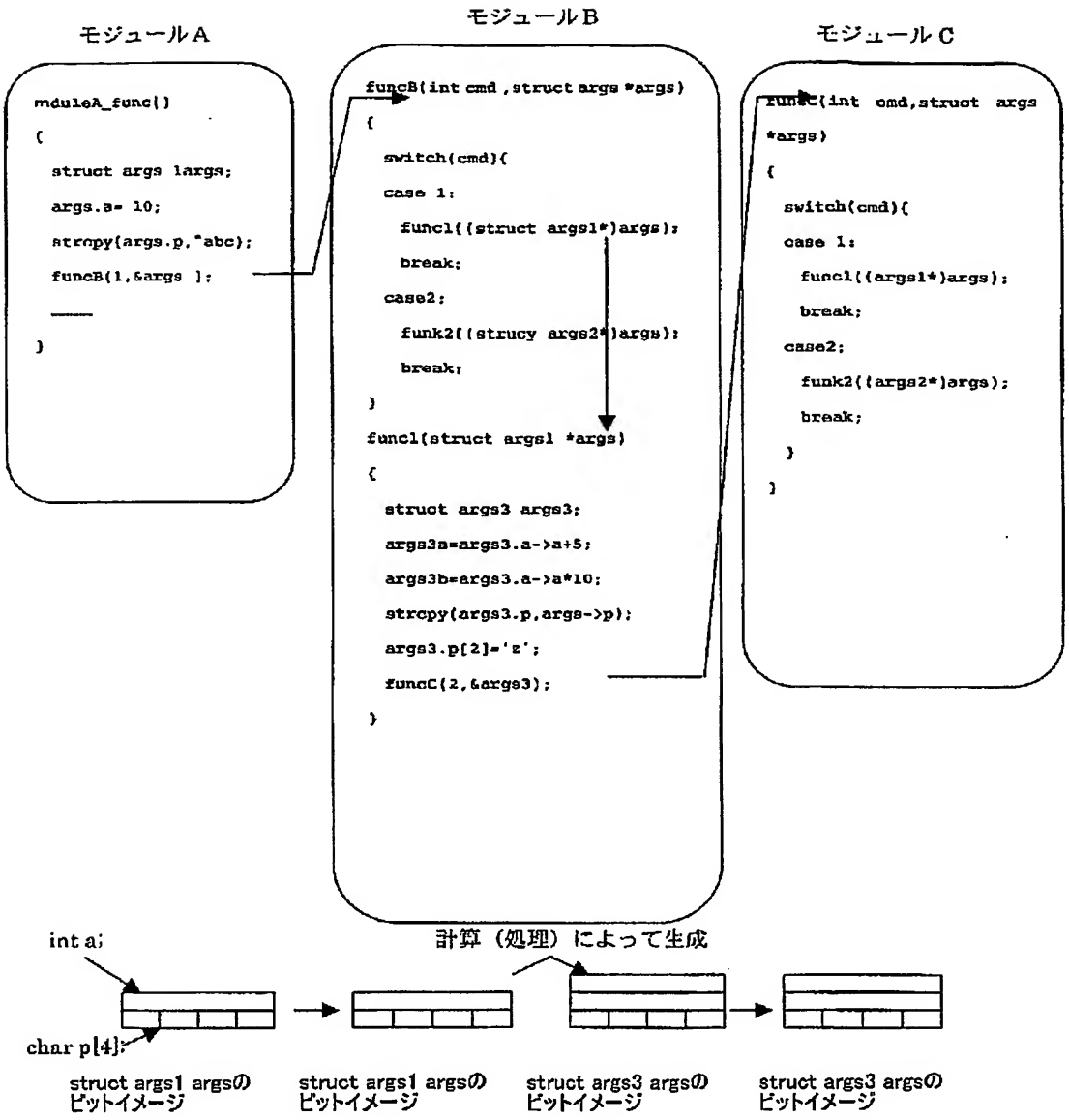
【図７】


モジュールA
関数呼び出し
モジュールB
struct args1 {
int a;
char p[4];
};
struct args2 {
int a ;
int b ;
int p [8] ;
} ;
Module_program( )
{
char *p = "abc";
struct args1 args;
args.a=3,
strcpy(args.p, p);
/* 変数のポインタを引数として呼び出す */
func(&args);
.....
}
func1 (struct args1 *args)
{
/* 構造体の内容を変数に戻す */
int a = args->a;
char *p = args->p;
char b;
b=p[a] + 3;
.....
return b;
}
func2 (struct args2 *args)
{
/* 構造体の内容を変数に戻す */
int a = args->a;
int b = args->b;
int *p = args->p;
int c;
c=p[a] * b;
.....
}

【図８】

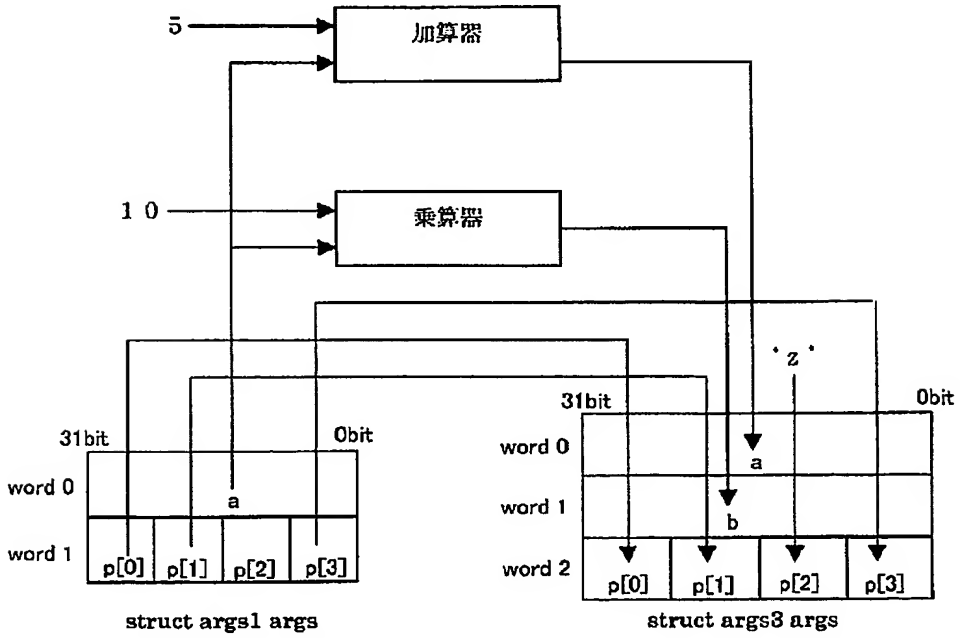

モジュールA
関数呼び出し
モジュールB
struct args {
int a;
char p[4]
};
struct args2 {
int a ;
int b ;
int p[8] ;
};
module_program( )
{
char *p = "abc";
struct args1 args ;
args.a=3,
strcpy(args.p, p);
/* 変数のポインタを引数として呼び出す */
func(1,(void*)(&args));
.....
}
func(int func ID, void *args)
{
switch(func ID) {
case 1:
func1(struct args1*) args
break;
case2;
func2(struct args2*) args
break;
}
}
func1 (struct args1 *args)
{
/* 構造体の内容を変数に戻す */
int a = args->a;
char *p = args->p;
char b;
b=p[a] + 3;
.....
return b;
}
func2 (struct args2*args)
{
/* 構造体の内容を変数に戻す */
int a = args->a;
int b = args->b;
int *p = args->p;
int c;
c=p[a] * b;
.....
}

【図９】

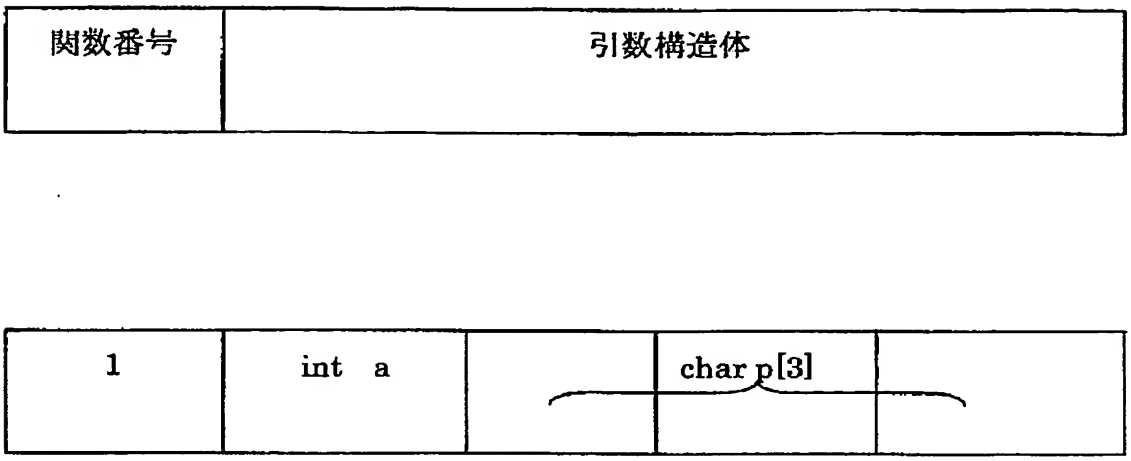

モジュールA
mduleA_func()
{
struct args 1args;
args.a= 10;
strcpy(args.p,"abc);
funcB(1,&args);
}
モジュールB
funcB(int cmd ,struct args *args)
{
switch(cmd){
case 1:
funcl((struct args1*)args);
break;
case2:
funk2((strucy args2*)args);
break;
}
funcl(struct args1 *args)
{
struct args3 args3;
args3a=args3.a->a+5;
args3b=args3.a->a*10;
strcpy(args3.p,args->p);
args3.p[2]='z';
funcC(2,&args3);
}
モジュールC
funcC(int omd,struct args *args)
{
switch(cmd){
case 1:
funcl((args1*)args);
break;
case2;
funk2((args2*)args);
break;
}
}
int a;
char p[4];
計算（処理）によって生成
struct args1 argsのビットイメージ
struct args1 argsのビットイメージ
struct args3 argsのビットイメージ
struct args3 argsのビットイメージ

【図１０】

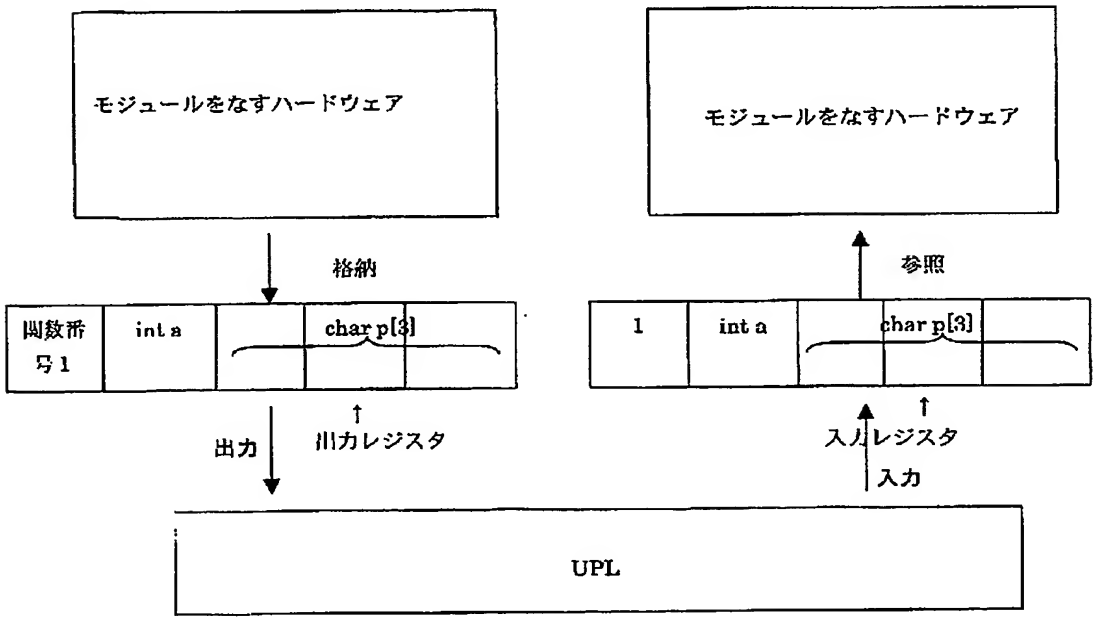
```
func1(struct args1 *args)
{
     struct args3 args3;
     args3.a = args->a+5;
     args3.b = args->a*10;
     strcpy(args3.p,args->p);
     args3.p[2] = 'z';
     funcC( 1 ,(void*)(&args3));
}
```



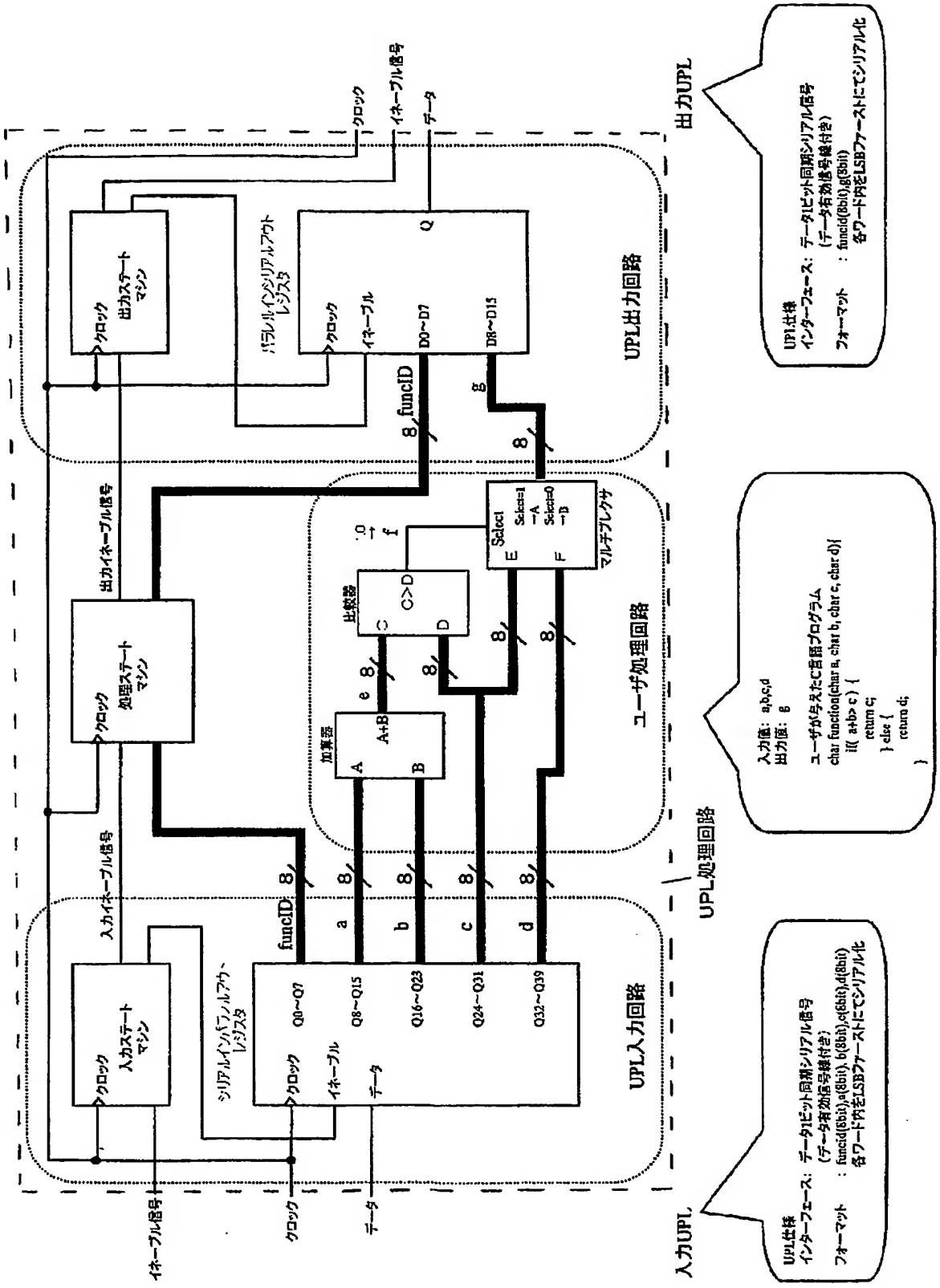
【図１１】


関数番号
引数構造体
1
int a
char p[3]

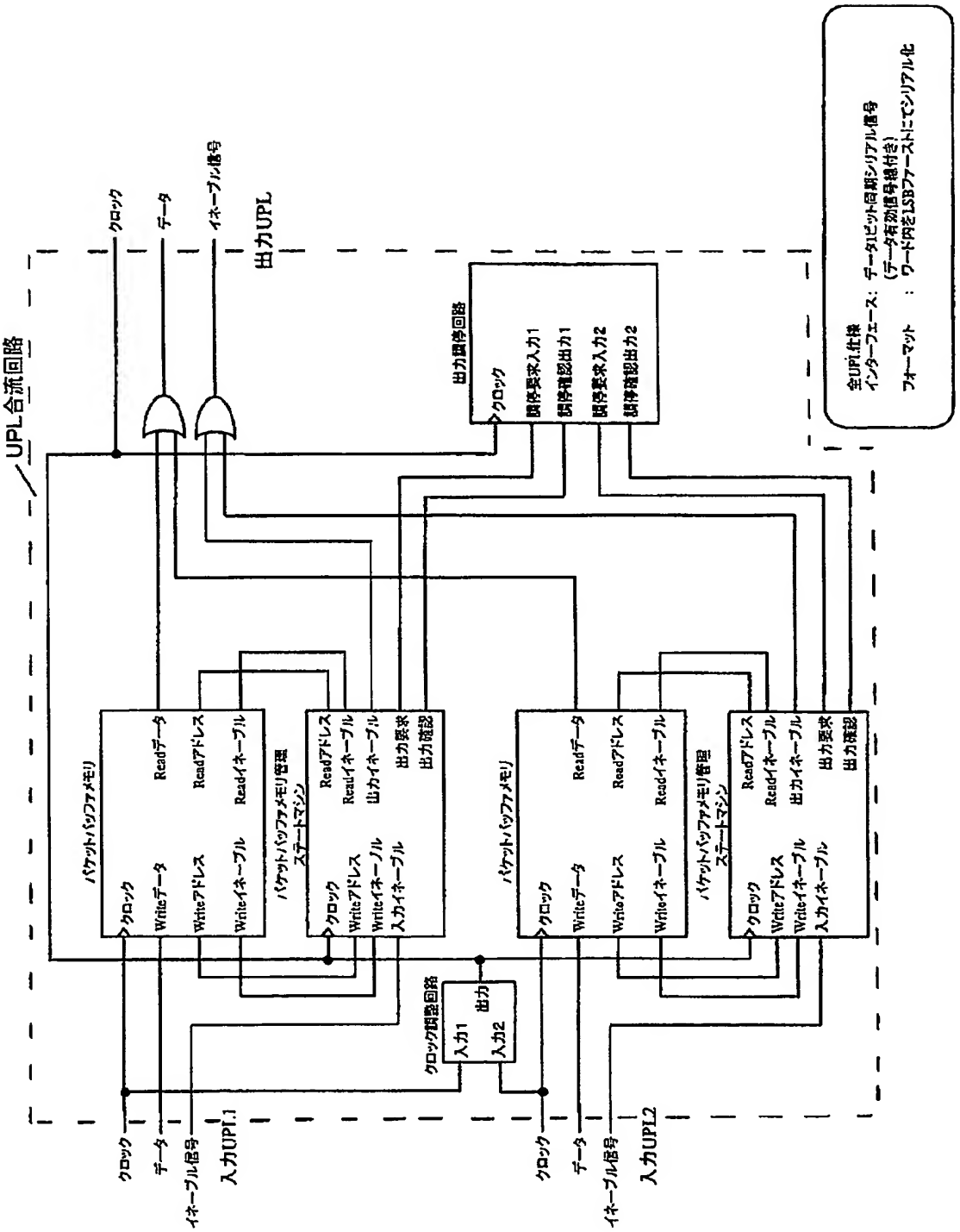
【図１２】


モジュールをなすハードウェア
モジュールをなすハードウェア
格納
参照
関数番号1
int a
char p[3]
1
int a
char p[3]
出力
出力レジスタ
入力レジスタ
入力
UPL

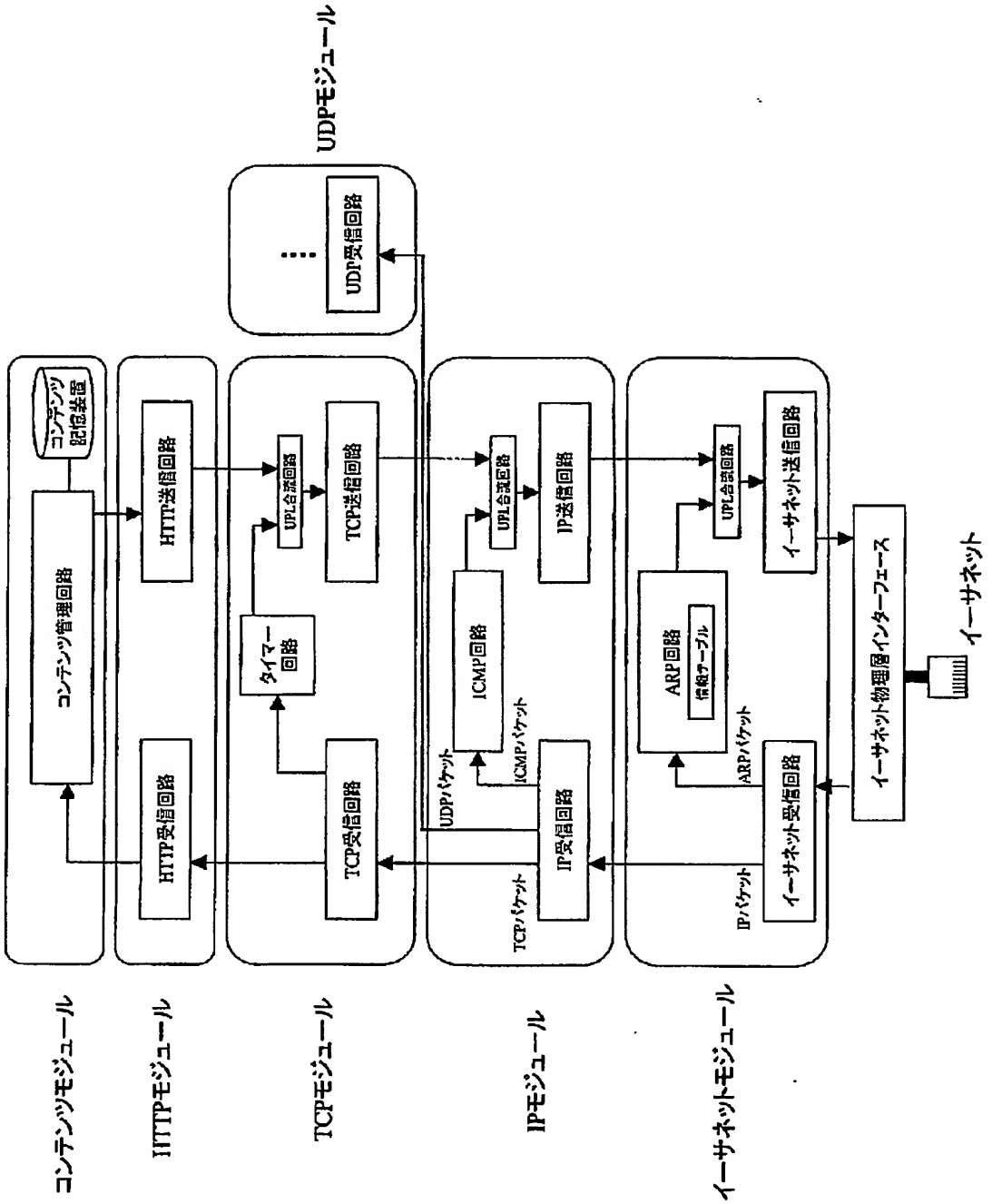
【図１３】


出力UPL
UPL出力回路
UPL処理回路
ユーザ処理回路
UPL入力回路
入力UPL
クロック
イネーブル信号
データ
出力ステートマシン
パラレルインシリアルアウトレジスタ
funcID
処理ステートマシン
入力ステートマシン
シリアルインパラレルアウトレジスタ
加算器
比較器
マルチプレクサ
Select
入力値: a,b,c,d
出力値: g
ユーザが与えたC言語プログラム
char function(char a, char b, char c, char d){
if(a+b> c) {
return c;
} else {
return d;
}

【図１４】

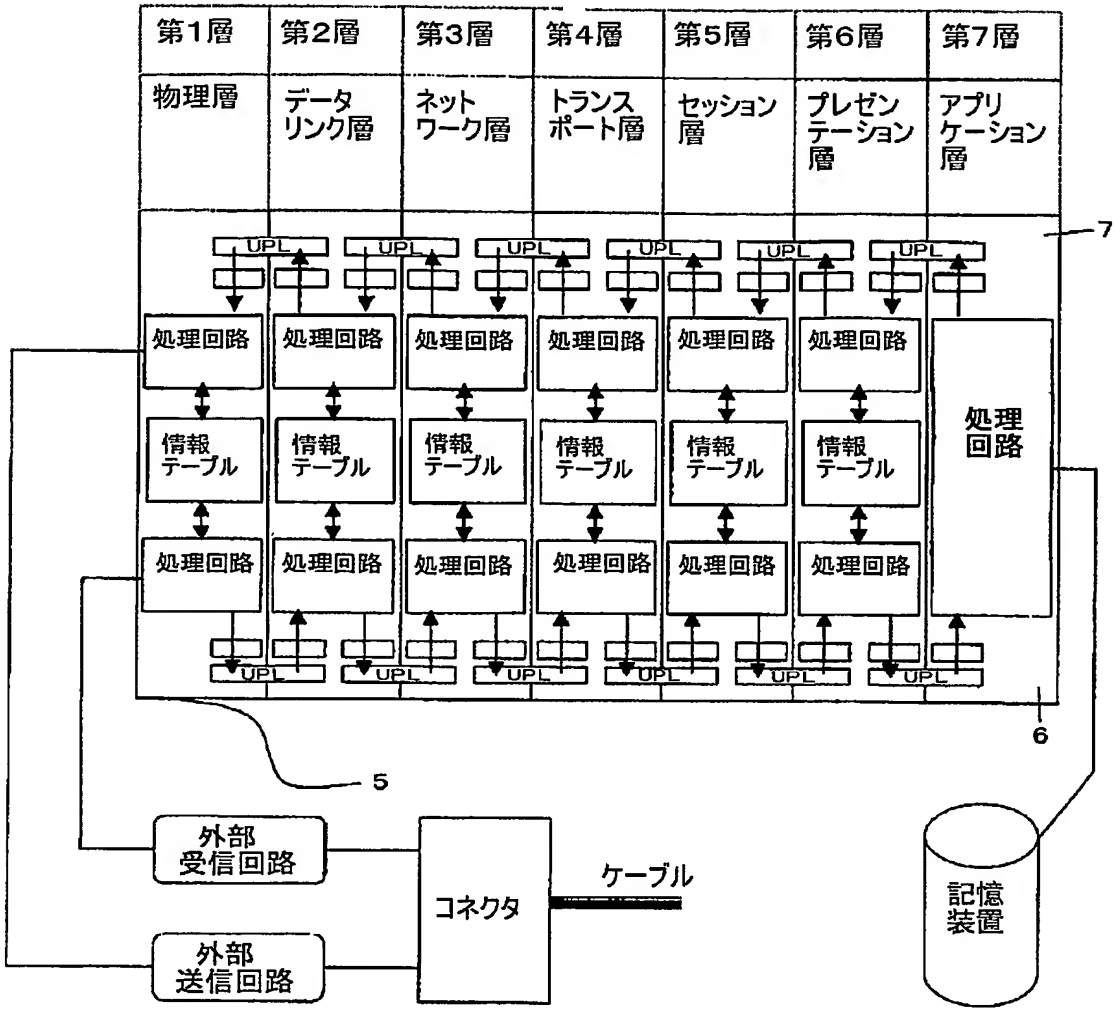

UPL合流回路
出力UPL
クロック
データ
イネーブル信号
出力調停回路
クロック
調停要求入力1
調停確認出力1
調停要求入力2
調停確認出力2
全UPI.仕様
インターフェース：データ1ビット同期シリアル信号
(データ有効信号組付き)
フォーマット　：ワード内をLSBファーストにてシリアル化
パケットバッファメモリ
クロック
Writeデータ
Writeアドレス
Writeイネーブル
Readデータ
Readアドレス
Readイネーブル
パケットバッファメモリ管理
ステートマシン
クロック
Writeアドレス
Writeイネーブル
入力イネーブル
Readアドレス
Readイネーブル
出力イネーブル
出力要求
出力確認
クロック調整回路
入力1
入力2
出力
入力UPI.1
入力UPL2

【図１５】

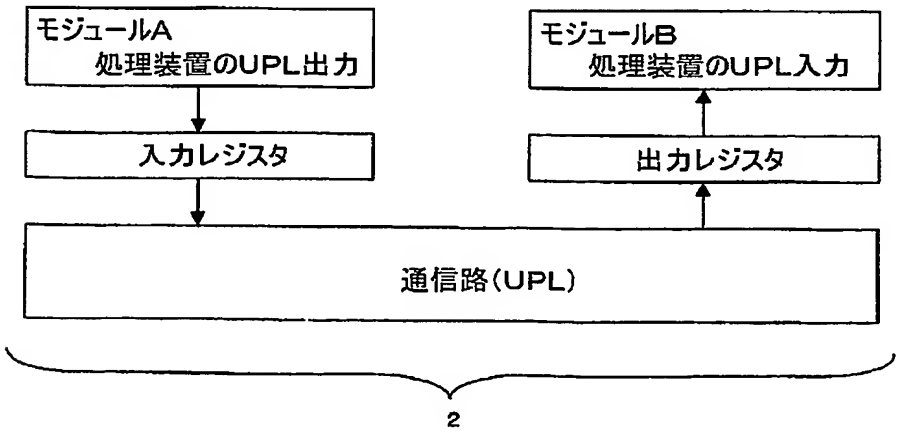

UDPモジュール
UDP受信回路
コンテンツモジュール
コンテンツ記憶装置
コンテンツ管理回路
HTTPモジュール
HTTP送信回路
HTTP受信回路
TCPモジュール
UPL合流回路
TCP送信回路
タイマー回路
TCP受信回路
IPモジュール
UPL合流回路
IP送信回路
ICMP回路
UDPパケット
ICMPパケット
IP受信回路
TCPパケット
イーサネットモジュール
UPL合流回路
イーサネット送信回路
ARP回路
情報テーブル
ARPパケット
イーサネット受信回路
IPパケット
イーサネット物理層インターフェース
イーサネット

【図１６】

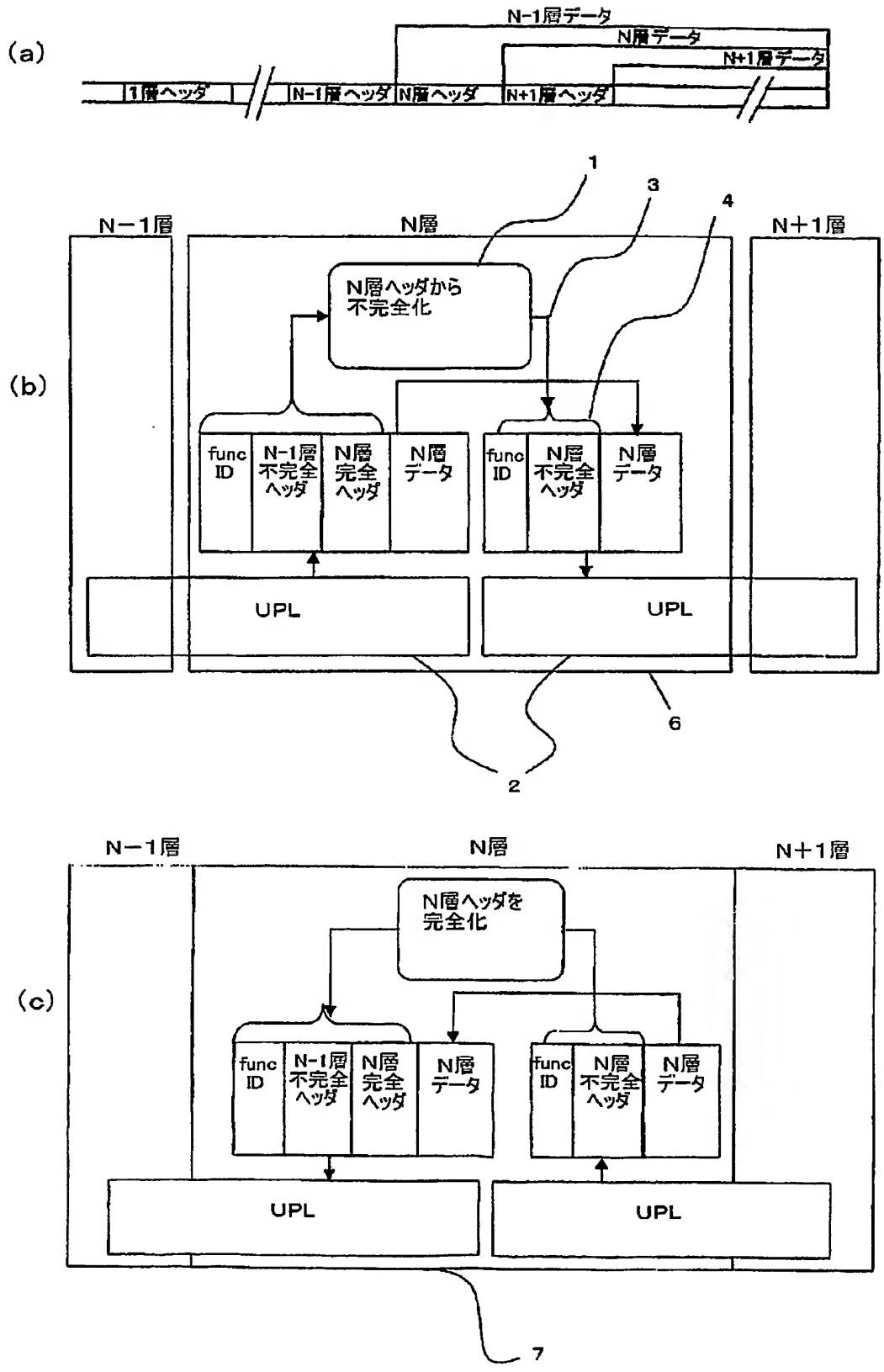
(a)


第1層
第2層
第3層
第4層
第5層
第6層
第7層
物理層
データリンク層
ネットワーク層
トランスポート層
セッション層
プレゼンテーション層
アプリケーション層
UPL
処理回路
情報テーブル
処理回路
処理回路
7
6
5
外部受信回路
外部送信回路
コネクタ
ケーブル
記憶装置

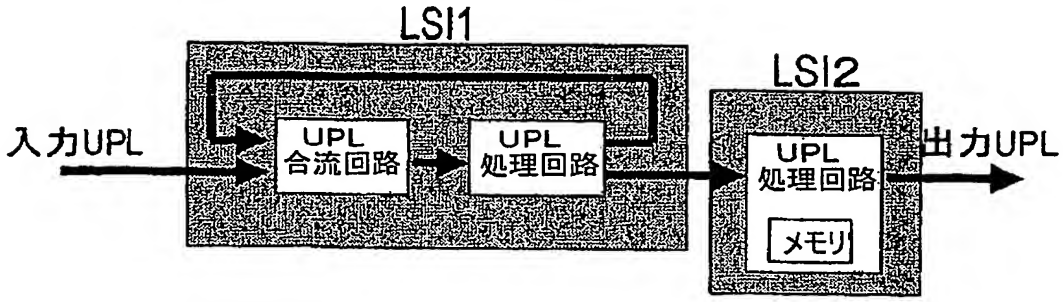
(b)


モジュールA
処理装置のUPL出力
モジュールB
処理装置のUPL入力
入力レジスタ
出力レジスタ
通信路(UPL)
2

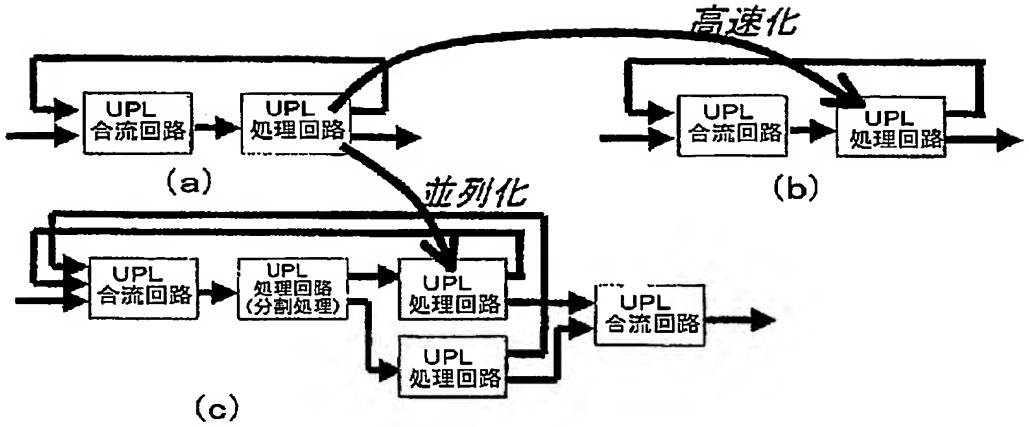
【図１７】


(a)
N-1層データ
N層データ
N+1層データ
1層ヘッダ
N-1層ヘッダ
N層ヘッダ
N+1層ヘッダ
(b)
N−1層
N層
N+1層
1
3
4
N層ヘッダから
不完全化
func ID
N-1層 不完全 ヘッダ
N層 完全 ヘッダ
N層 データ
func ID
N層 不完全 ヘッダ
N層 データ
UPL
UPL
6
2
(c)
N−1層
N層
N+1層
N層ヘッダを
完全化
func ID
N-1層 不完全 ヘッダ
N層 完全 ヘッダ
N層 データ
func ID
N層 不完全 ヘッダ
N層 データ
UPL
UPL
7

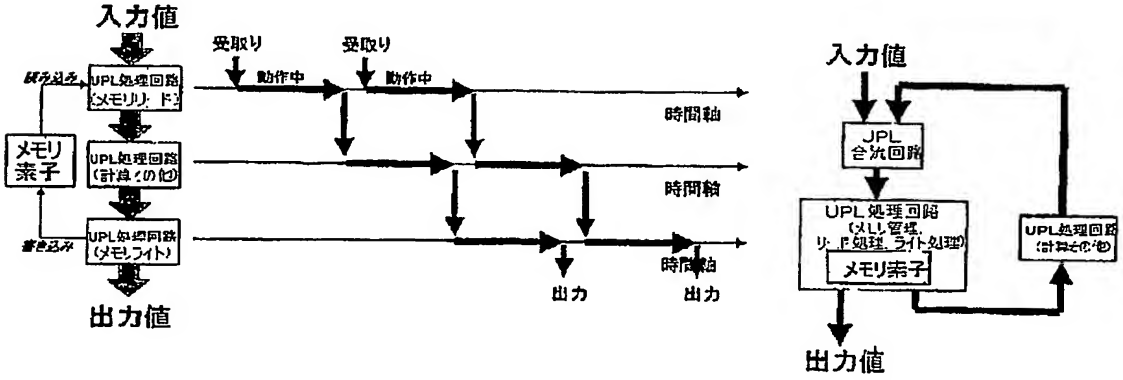
【図１８】


LSI1
LSI2
入力UPL
UPL
合流回路
UPL
処理回路
UPL
処理回路
メモリ
出力UPL

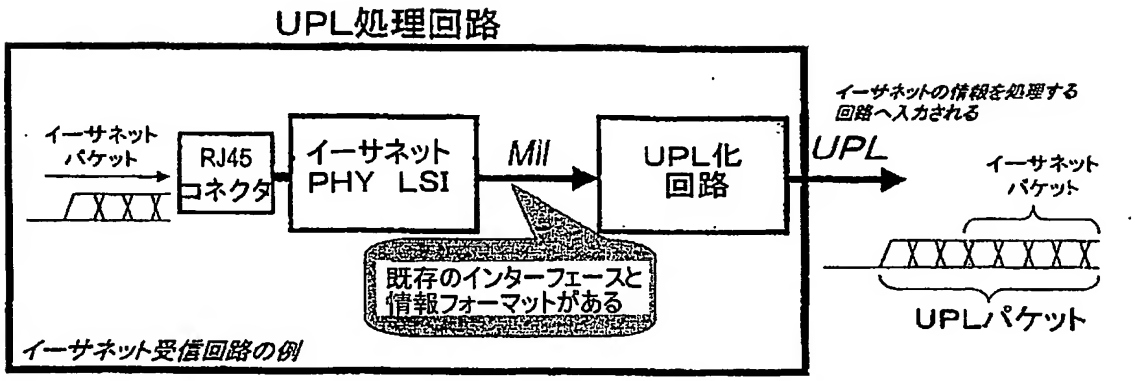
【図１９】


高速化
UPL
合流回路
UPL
処理回路
(a)
UPL
合流回路
UPL
処理回路
(b)
並列化
UPL
合流回路
UPL
処理回路
(分割処理)
UPL
処理回路
UPL
処理回路
UPL
合流回路
(c)

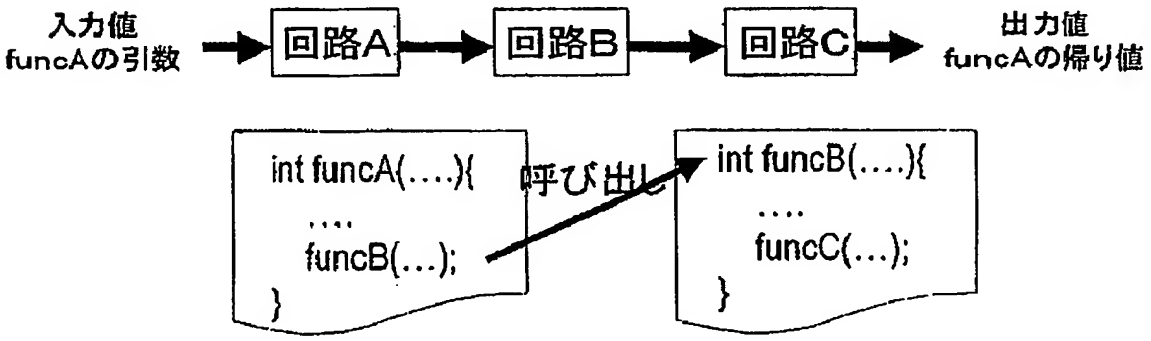
【図２０】


入力値
受取り
受取り
動作中
動作中
時間軸
時間軸
時間軸
メモリ
素子
出力
出力
出力値
入力値
JPL
合次回路
メモリ素子
出力値

【図２１】

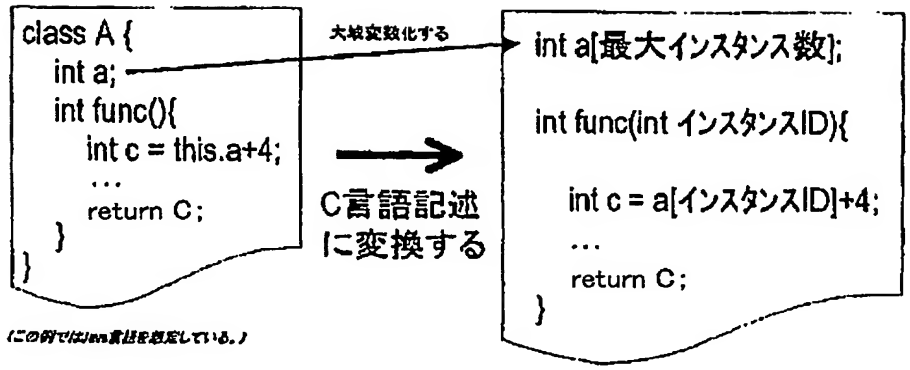

UPL処理回路
イーサネット
パケット
RJ45
コネクタ
イーサネット
PHY LSI
MII
UPL化
回路
UPL
イーサネットの情報を処理する
回路へ入力される
イーサネット
パケット
UPLパケット
既存のインターフェースと
情報フォーマットがある
イーサネット受信回路の例

【図２２】


入力値
funcAの引数
回路A
回路B
回路C
出力値
funcAの帰り値
int funcA(....){
....
funcB(...);
}
呼び出し
int funcB(....){
....
funcC(...);
}

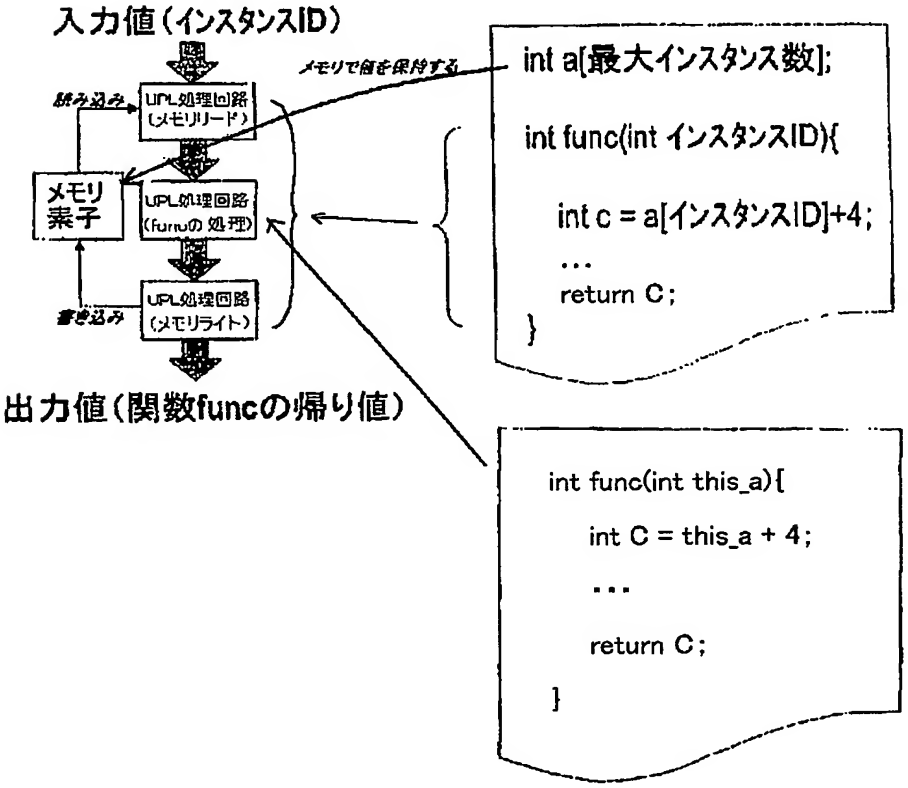
【図２３】

(a)

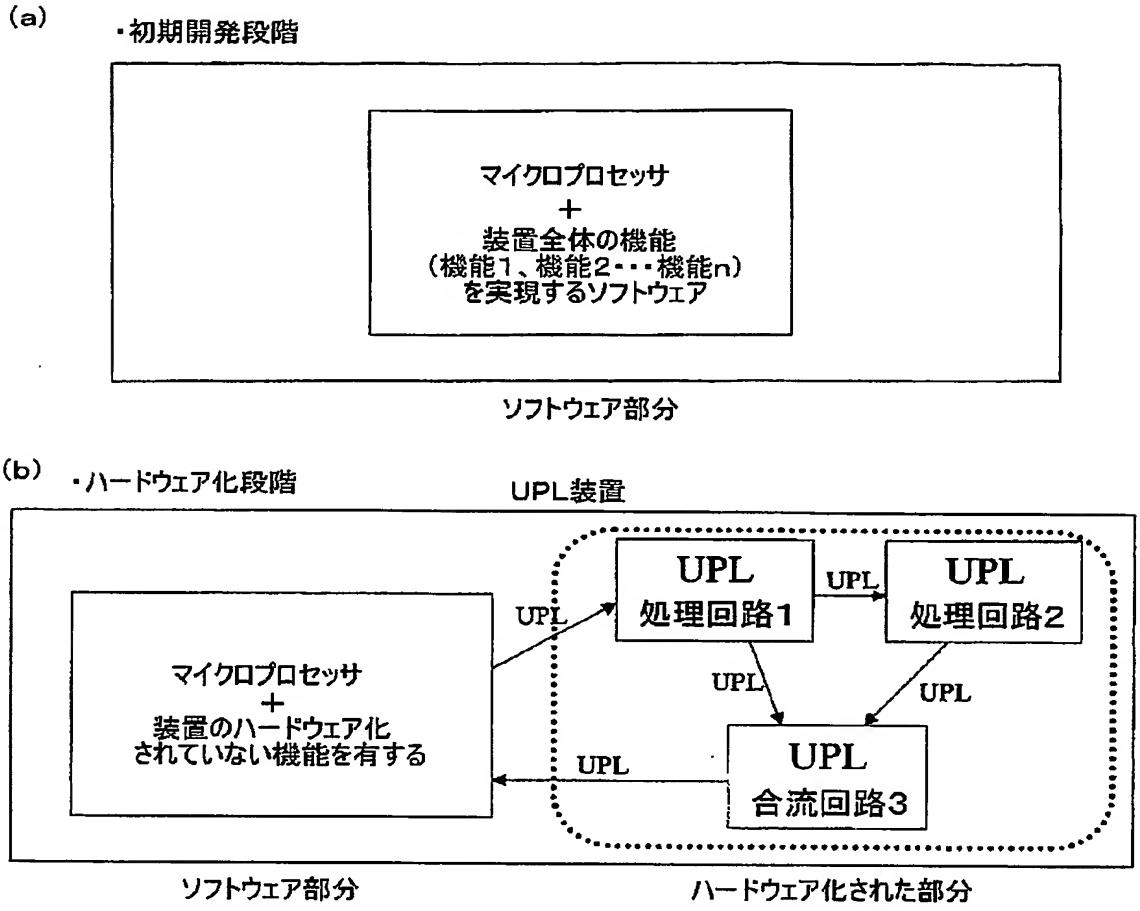


- クラス変数やインスタンス変数を大域変数へと変換する。
- クラスのインスタンスを指し示すインスタンスIDを関数呼び出し時に指定する。

(b)



【図２４】


(a)
・初期開発段階
マイクロプロセッサ
＋
装置全体の機能
（機能1、機能2・・・機能n）
を実現するソフトウェア
ソフトウェア部分
(b)
・ハードウェア化段階
UPL装置
マイクロプロセッサ
＋
装置のハードウェア化
されていない機能を有する
UPL
UPL処理回路1
UPL処理回路2
UPL合流回路3
UPL
ソフトウェア部分
ハードウェア化された部分
(c)
・開発終了段階
UPL装置
マイクロプロセッサ
＋
装置のハードウェア化
されない機能を有する
UPL処理回路1
UPL処理回路2
UPL処理回路3
UPL合流回路4
UPL処理回路5
UPL合流回路6
UPL
ソフトウェア部分
ハードウェア部分

【書類名】　要約書

【要約】

【課題】　本発明は、情報の伝達および計算処理、特にインターネットサーバ機能をハードウェアモジュールによって実現する方法および装置を提供する。

【解決手段】　本発明のハードウェアモジュールによる情報処理装置は、基本的に、２種類の回路モジュール、即ち、１もしくは複数の処理回路モジュールと、０もしくは１以上の合流回路モジュールとから構成される。前記処理回路モジュールおよび前記合流回路モジュールは、所定の情報を含むパケットによる単一方向の情報伝達を、前記処理回路モジュール、前記合流回路モジュールまたはＩ／Ｏインターフェースとの間で行う。また、前記処理回路モジュールの各々は、そのモジュール固有の機能を果たす回路と、０または１の入力と、０以上の出力とを有している。さらに、前記合流回路モジュールは、２つ以上の入力と１つの出力のみを有し、２つ以上の回路から出力されるパケットを１つの出力に合流する機能をもつ。

本発明の装置においては、前記回路モジュール間における信号の伝達がパケット単位で行われるようになっている。そして、回路モジュールには、動作に必要な値を含むパケットが入力され、回路モジュール内で所定の処理を行い、結果の値を含むパケットを出力する。このように、回路モジュール間の情報伝達がパケットによって行われるため、設計者は複雑な入出力のタイミングの制御という課題から開放される。さらに、パケット単位でのパイプライン処理が可能であるので、回路モジュールを含む装置に与えられた複数の処理を効率的に行い、全体的な処理速度の向上を達成することができる。また、ＵＰＬ装置はデバック作業等が容易なため、従来より低コスト、短期間で装置開発が可能である。

【選択図】　図４

認定・付加情報

特許出願の番号	特願２００２－１８６４７８
受付番号	５０２００９３６４９１
書類名	特許願
担当官	第七担当上席　　００９６
作成日	平成１４年　６月２７日

＜認定情報・付加情報＞

【提出日】　平成14年 6月26日

次頁無

特願２００２－１８６４７８

出　願　人　履　歴　情　報

識別番号　　［５０１０５３０２６］

１．変更年月日　　２００２年　４月　１日
［変更理由］　　名称変更
住所変更
住　所　　東京都渋谷区広尾１丁目１１番４号　共立ビル５０１
氏　名　　株式会社イーツリーズ・ジャパン

This Page is Inserted by IFW Indexing and Scanning Operations and is not part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☐ FADED TEXT OR DRAWING
- ☑ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☐ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.
As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.